1. Project Documentation (For Client) 2
1.1 1 Project Overview 3
1.2 2 Requirements and Scope 4
1.2.1 2.1 Background 5
1.2.2 2.2 Functional Requirements 11
1.2.3 2.3 Non-Functional Requirements 12
1.2.4 2.4 Project Scope 13
1.2.5 2.5 Risk Management 14
1.2.6 2.6 Motivational Model 15
1.2.7 2.7 User Personas 16
1.2.8 2.8 User Stories 17
1.2.9 2.9 Acceptance Criteria 18
1.3 3 Architecture Design 19
1.3.1 2.1 Architecture Representation 20
1.3.2 2.2 Software Architecture 22
1.4 4 Use Cases 24
1.4.1 4.1 Actors 25
1.4.2 4.2 Use Cases 26
1.5 5 Project Implementation 28
1.5.1 5.1 Platform Implementation 31
1.5.2 5.2 Front-End Implementation 32
1.5.3 5.3 Back-End Implementation 43
1.5.4 5.4 Jupyter Implementation 53
1.5.5 5.5 Database Implementation 54
1.5.5.1 5.5.1 Overview 55
1.5.5.2 5.5.2 Entity Relationship Diagram 57
1.5.5.3 5.5.3 Data Dictionary 58
1.6 6 Tutorials 73
1.6.1 6.1 Data Ingestion Tutorial 74
1.6.2 6.2 Data Preparation Tutorial 87
1.6.3 6.3 Jupyter Tutorial 94
1.6.4 6.4 Software (Platform) Deployment & Maintenance Tutorial 100
1.7 7 Testing 101
1.7.1 7.1 Acceptance Testing 102
1.7.2 7.2 Unit Testing 112
1.8 8 Future Considerations 113

Project Documentation (For Client)

This section contains the finalized project documentation that will be sent to the clients. It contains documentation on our system's architecture and guides to setting up and executing the system.

- 1 Project Overview
- 2 Requirements and Scope
- 3 Architecture Design
- 4 Use Cases
- 5 Project Implementation
- 6 Tutorials
- 7 Testing
- 8 Future Considerations

1 Project Overview

Project Name: Automated Audit Benchmarking

Project Team Name: AA-Wombat

Project Background and Description:

This project was proposed by the Australian Clinical Dosimetry Service (ACDS) from the Australian Radiation Protection and Nuclear Safety Agency (ARPANSA) to develop a program that can automatically report a benchmark of an audit against a national data set. Each audit involves measuring the dose variation of the output of a linear accelerator in radiation oncology facilities. This data collected by the ACDS is compared with the national data set and a benchmark report is generated and provided to each audit client so that they can maintain quality assurance in their treatment facilities.

Currently, the process to generate a benchmark report consists of manually inputting audit data into an Excel spreadsheet and producing a chart that compares each an audit result with the national data set. Recently, the Excel spreadsheet has become unwieldy and slow due to the amount of data from accumulated audits and the ACDS are looking into replacing it with a new system. The aim of this project is to develop a program that can replace the usage of the Excel spreadsheet and create audit charts with a quicker and more efficient process.

Vision Statement:

The ultimate vision of this project is to have an automation system which allows audits to be processed in an efficient, fast and cost effective way. The project will seek to ensure that audit processing will not be a heavy burden on the scientists and will be robust to changes and consistent in its output.

Team Members:

Member Name	Student Number	Mobile Number	Student Email	Role
David Tran	587691	0479169238	davidt@student.unimelb.edu.au	Engineer
Benedict Ong	761314	0434728846	benedicto@student.unimelb.edu.au	Scrum Master
Samuel Jones	990929	0478545788	stjones@student.unimelb.edu.au	Product Owner
Ariel Kark	521428	0406815663	akark@student.unimelb.edu.au	Engineer

Supervisor:

Supervisor Name	Email
Paul Calverley	pcalverley@unimelb.edu.au

Client Details:

Organization Name: Australian Radiation Protection and Nuclear Safety Agency (ARPANSA)

Client Name	Mobile Number	Email
Sabeena Beveridge	0452622232	Sabeena.Beveridge@arpansa.gov.au
Andrew Alves	0401796886	Andrew.Alves@arpansa.gov.au
Maddison Shaw	0439087405	Maddison.Shaw@arpansa.gov.au

Useful Links for the Client:

- Google Drive: https://drive.google.com/drive/u/2/folders/1kPJBe_kChQ_qDEtenha08i54zNhFLEB0
- Github: https://github.com/COMP90082-2020-SM2-AA-Wombat

2 Requirements and Scope

This section describes the requirements of the system. The requirements analysis defines the scope boundaries and includes preliminary user stories are demonstrated as a broad overview of the system requirements. The requirements and tasks are managed in Jira.

Link to Jira: https://jira.cis.unimelb.edu.au:8444/projects/AAWO8220/summary

For further analysis of the requirements.

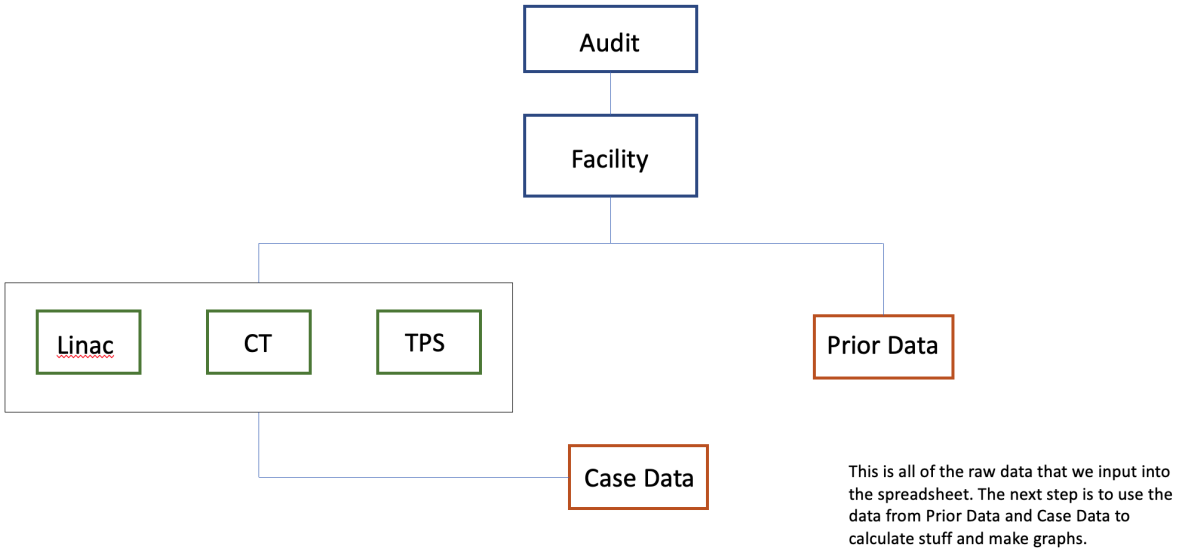
- 2.1 Background
- 2.2 Functional Requirements
- 2.3 Non-Functional Requirements
- 2.4 Project Scope
- 2.5 Risk Management
- 2.6 Motivational Model
- 2.7 User Personas
- 2.8 User Stories
- 2.9 Acceptance Criteria

2.1 Background

This project was proposed by the Australian Clinical Dosimetry Service (ACDS) from the Australian Radiation Protection and Nuclear Safety Agency (ARPANSA) to develop a program that can automatically report a benchmark of an audit against a national data set. Each audit involves measuring the dose variation of the output of a linear accelerator in radiation oncology facilities. This data collected by the ACDS is compared with the national data set and a benchmark report is generated and provided to each audit client so that they can maintain quality assurance in their treatment facilities.

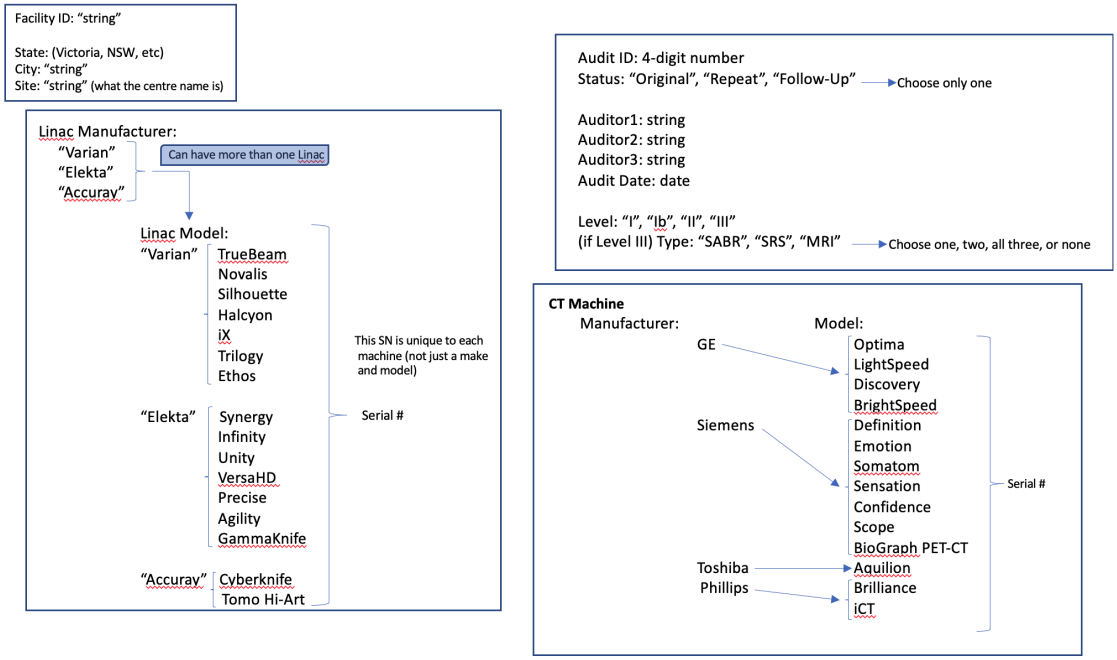
Currently, the process to generate a benchmark report consists of manually inputting audit data into an Excel spreadsheet and producing a chart that compares each an audit result with the national data set. Recently, the Excel spreadsheet has become unwieldy and slow due to the amount of data from accumulated audits and the ACDS are looking into replacing it with a new system. The aim of this project is to develop a program that can replace the usage of the Excel spreadsheet and create audit charts with a quicker and more efficient process.

Information about current process provided by the client:



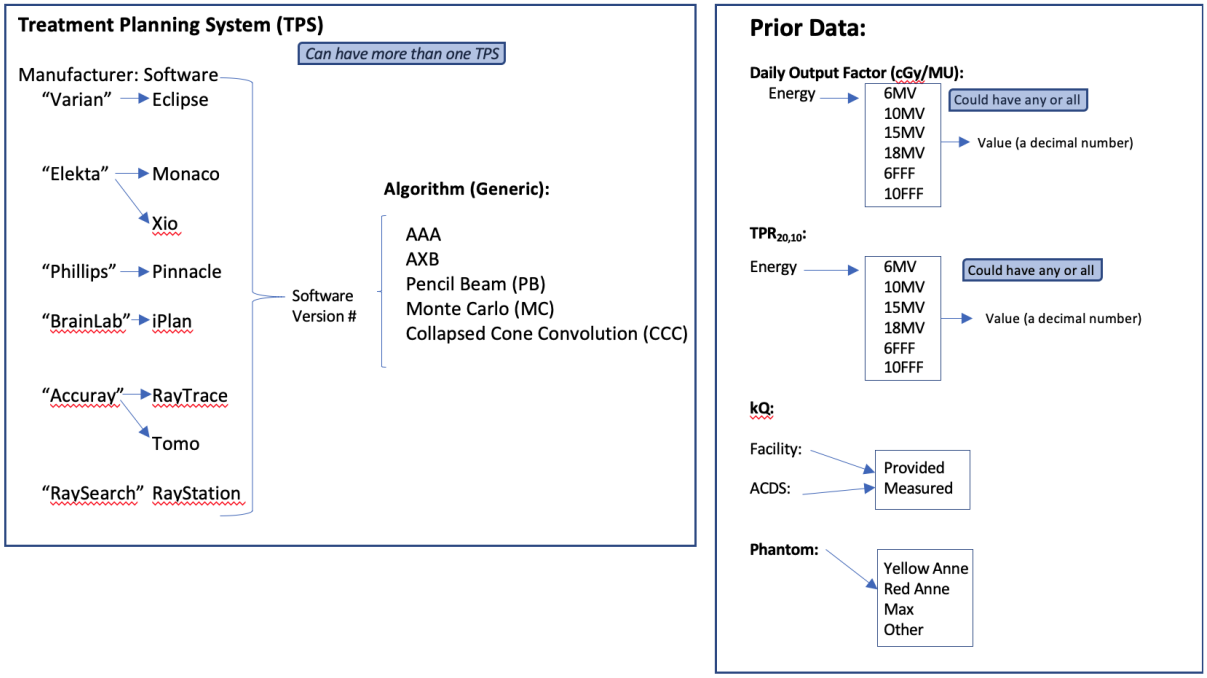
Slide 1.

Audits occur on **facilities** and have levels - we are only looking at level III but there are other levels. Audits are identified by both an audit number but also have a version number. Since the reports generated from these audits have large impacts on a facilities ability to provide treatment (i.e. the facility may be temporarily closed) if the audit results are within a threshold - or if perhaps weren't performed correctly, the audit will be performed again with a different version. So audit results are uniquely identified by an audit number AND a a version number .



Potential data values for Linac and CT machines Audits and Facilities NB: Audit should also include a version number not shown in this chart.

Linac, CT, TPS corresponds to rows Q - AG in the results tab of spreadsheet. All facilities have some combinations of these that are used in a treatment plan. Flow = Client gets CT Scan, which is used to create a treatment plan on a TPS which is used to deliver a treatment through a Linac machine. We are mainly looking at the results from he linac but need to record information about the CT and TPS machines used in the process.



The **prior data** is produced by a facility on the day of the audit and corresponds to Rows AH to AV of the results tab of the spreadsheet. This table of data needs to be linked uniquely to an audit and version performed on a particular day and be linkable to case results.

This contains information about daily output (how "hot" the machine is running that day), and TPR 10,20, and Kq values - these are basically values that are provided by the facility from the Linac machine that are used to calibrate the machine for a treatment. For example if the machine is a little hot they may lower the energy for a dose by say 0.5%. This information is often referenced in relation to case data to see if a facility is Offsetting its treatments correctly.

AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW
										last row	264				
Facility Stated															
Daily Output (cGy/MU)						$TPR_{20,10}$						k_Q			
6	10	15	18	6FFF	10FFF	6	10	15	18	6FFF	10FFF	Fac.	ACDS	Phantom	
0.999	1.002	#N/A	#N/A	#N/A	#N/A	0.661	0.735	#N/A	#N/A	#N/A	#N/A	m	m		
1.000	1.001	#N/A	#N/A	#N/A	#N/A	0.667	0.740	#N/A	#N/A	#N/A	#N/A	p	m		
0.998	0.999	#N/A	#N/A	0.999	#N/A	0.669	0.735	#N/A	#N/A	0.633	#N/A	m	m	Yellow Anne	
1.013	1.010	#N/A	#N/A	1.012	#N/A	0.666	0.630	#N/A	#N/A	0.740	#N/A	m	m	Yellow Anne	
0.999	1.002	#N/A	#N/A	#N/A	#N/A	0.661	0.735	#N/A	#N/A	#N/A	#N/A	m	m		
1.001	1.000	#N/A	#N/A	1.002	#N/A	0.664	0.741	#N/A	#N/A	0.628	#N/A	m	m	Yellow Anne	
0.998	0.992	#N/A	#N/A	0.991	0.989	0.665	0.738	#N/A	#N/A	0.631	0.704	m	m	Yellow Anne	
0.998	0.992	#N/A	#N/A	#N/A	#N/A	0.665	0.738	#N/A	#N/A	#N/A	#N/A	m	m	Yellow Anne	
#N/A	#N/A	#N/A	#N/A	1.004	#N/A	#N/A	#N/A	#N/A	#N/A	0.704	#N/A	m	m		
1.012	1.011	#N/A	#N/A	1.011	1.012	0.667	0.739	#N/A	#N/A	0.629	0.705	p	m		
1.012	1.011	#N/A	#N/A	#N/A	#N/A	0.667	0.739	#N/A	#N/A	#N/A	#N/A	p	m		
#N/A	#N/A	#N/A	#N/A	1.002	1.011	#N/A	#N/A	#N/A	#N/A	0.629	0.708	p	m	Max	
1.008	#N/A	#N/A	#N/A	1.007	#N/A	0.664	#N/A	#N/A	#N/A	0.628	#N/A	m	m	Max	
1.008	#N/A	#N/A	#N/A	1.007	#N/A	0.664	#N/A	#N/A	#N/A	0.628	#N/A	m	m	Max	
1.002	#N/A	#N/A	#N/A	1.002	1.005	0.668	#N/A	#N/A	#N/A	0.632	0.708	m	m	Max	
1.002	#N/A	#N/A	#N/A	1.002	1.005	0.668	#N/A	#N/A	#N/A	0.632	0.708	m	m	Max	
1.002	#N/A	#N/A	#N/A	1.002	1.005	0.668	#N/A	#N/A	#N/A	0.632	0.708	m	m	Max	
#N/A	#N/A	#N/A	#N/A	1.008	#N/A	#N/A	#N/A	#N/A	#N/A	0.635	#N/A	m	m	Max	
#N/A	#N/A	#N/A	#N/A	1.008	#N/A	#N/A	#N/A	#N/A	#N/A	0.635	#N/A	m	m	Max	
#N/A	#N/A	#N/A	#N/A	0.991	#N/A	#N/A	#N/A	#N/A	#N/A	0.630	#N/A	m	m	Max	
#N/A	#N/A	#N/A	#N/A	0.991	#N/A	#N/A	#N/A	#N/A	#N/A	0.630	#N/A	m	m	Max	
0.996	1.000	#N/A	#N/A	0.999	#N/A	0.671	0.737	#N/A	#N/A	0.629	#N/A	m	m	Yellow Anne	
1.002	1.005	#N/A	#N/A	1.005	#N/A	0.680	0.740	#N/A	#N/A	0.680	#N/A	m	m	Yellow Anne	

Case Data

These are the measurements taken from the Phantom. Each case is a combination of a beam or beams measured at specific point in the phantom. You can think if each point as a specific place in the phantom that you can place a sensor to measure the energy. The points always refer to the same place on any phantom. The cases may be delivered at any energy level. (energy levels are discrete , defined, ordinal values) an audit may include the same case at different energy levels. The RLAT, LLAT tags are just for human reading and can be ignored (they are the beam direction e.g. Right Lateral) Av, StdDev and N are calculated values. In addition to measured value our database should have the ability to optionally store a corresponding predicted value.

AX	AY	AZ	BA	BB	BC
			RLAT	RLAT	AP
Case	1	1	2	2	2
Point	1	10	5	8	5
Beam	1	1	1	1	2
Energy	6	6	6	6	6
av	-0.37%	-0.81%	-0.46%	-2.06%	-0.62%
stdev	1.1%	1.4%	2.2%	3.0%	1.1%
n	207	198	163	162	163
	101106	110106	205106	208106	20520(
	-0.9%	-2.2%	-0.9%	-1.9%	-1.5%
	1.0%	1.0%	-1.8%	-2.2%	0.4%
	0.2%	-1.5%	2.1%	-5.6%	0.6%
	-1.9%	-1.7%	-4.4%	-4.3%	-3.5%
	-0.9%	-2.2%	-0.9%	-1.9%	-1.5%
	-0.8%	-3.6%	0.9%	-7.0%	-2.6%
	-0.7%	-1.5%	-0.4%	1.6%	-0.3%
	0.8%	-2.0%	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	-1.9%	-2.5%	-2.9%	-0.5%	-0.8%
	-0.7%	-2.6%	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A
	-2.2%	-2.8%	-4.0%	-2.2%	-2.1%
	-0.8%	-2.1%	-1.8%	-0.5%	-1.3%
	#N/A	#N/A	#N/A	#N/A	#N/A
	0.4%	-0.2%	0.3%	0.5%	0.0%
	-1.2%	-0.6%	#N/A	#N/A	#N/A

	-1.4%	-1.8%	-2.5%	-0.5%	-1.4%
	-0.2%	-2.2%	#N/A	#N/A	#N/A
	0.6%	0.2%	1.4%	0.0%	0.3%
	-0.4%	-0.1%	#N/A	#N/A	#N/A
	-0.1%	-0.5%	-1.5%	-1.4%	0.5%
	#N/A	#N/A	#N/A	#N/A	#N/A
	#N/A	#N/A	#N/A	#N/A	#N/A

Case tree spread sheet from Sabeena with more information about case data:

Case Trees.xlsx

2.2 Functional Requirements

Architectural Significant Functional Requirements

Function	Description
Workflow	The analyst should be able to upload CSV files to a datastage to be ingested to relevant tables in the database.
System Management	Analysts should be able to store and manipulate data as a source of truth. They can access this data at any time with the most up to date values in order to query, perform calculations, or create Python libraries.
System Management	• Production servers hosting the database and solution will be hosted on machines as directed by the client. Their scaling will be at the client's discretion. • The database will be designed to contain all the required objects to store the required data for analysis. Any new objects after project completion will have to be created by the client.
Analysis	Analysts should be able to pull data from the database into an external environment. Data analysis can be achieved through an interface between an external environment and the database. This will allow database queries to be generated, along with charts and reports.
Licensing	All the software and database involving in this project should have open source licenses.
Hardware	ACDS will provide all the necessary hardware and storage services for the production environment in the implementation phase. In the case of the development environment, the University of Melbourne, and students of the AA-Wombat team will oversee the hardware and storage requirements.

2.3 Non-Functional Requirements

Architectural Significant Non-Functional Requirements

Function	Description
Usability	The focus of the project is to build an accessible database with GUI or another interface for insertion/retrieval of the data being a secondary objective.
Reliability	All the stored data should be accessible for the analysts to make queries, calculations, and calls within a built library in Python. These data might change in time, so the system should be able to present the latest value of the stored data.
Reliability	The analysis and development of the system will be subject to the data provided by the ACDS. Therefore, the functionality will be bounded by the information provided by them.
Performance	The database response should be able to provide higher performance and more efficient workflows compared to using the Excel spreadsheet.
Supportability	Analysts have experience with Python and Matlab; they also have experience working with the SQL database language, so the system should be built using these programming languages. Additionally, all system code must be well commented to support future modifications and continuous improvements.
Security	The system should provide a level of security to protect access and maintain integrity of the data by verifying the authorization of a user before they can add new or modify data..

2.4 Project Scope

Scope

<table>
<tr><th>Project Scope</th><th></th></tr>
<tr><td>In Scope</td><td>

Construct a new automated workflow system that can replace the current workflow using the Excel spreadsheet.
Develop a database that can store audit results and be accessed remotely.
Develop a program that can access data from the database and generate benchmark charts for each audit result.
The workflow using the new system should be more efficient than the current workflow using the Excel spreadsheet and not too complex so that the client's can understand how it works.
Documentation for the new system so that it can be updated and maintained by the ACDS.

</td></tr>
<tr><td>Out of Scope</td><td>

Populating the database with all the past data, a subset of the data to demonstrate the functionality would be sufficient.
Developing functionality to retrieve new information from the database that is difficult or cannot be analysed with the current Excel spreadsheet.
Deploying the new system at ARPANSA.
Improving the data collection process of the audits.
Creating the audit reports, the program only needs to be able to create charts.

</td></tr>
<tr><td>Constraints</td><td>

The clients have experience with Python and MATLAB so the project will be developed with these two programming languages preferably so that it can continue to be updated and maintained.
Similarly, they also have some experience with SQL in regards to database programming languages.
Due to advances in radiation measuring techniques, there needs to be a degree in flexibility in the new system to accommodate for new fields in the database.
There is no allocated funds or budget for this project but the client can purchase licences or seek funds if required.
Due to the COVID-19 situation, all meetings will be held online.
The project team will have to balance their time between their workplace, studies and time to dedicate to this project.
Time will be spent by team members to learn the relevant skills and technologies.
The clients will be busy and not be available to answer questions all the time.
The project will have the following schedule:

<table>
<tr><th>Milestone</th><th>Date</th></tr>
<tr><td>Project Inception</td><td>9/8/2020 – 30/8/2020</td></tr>
<tr><td>Sprint 1</td><td>31/8/2020 – 27/9/2020</td></tr>
<tr><td>Final Presentation</td><td>26/10/2020 – 30/10/2020</td></tr>
<tr><td>Sprint 2</td><td>28/9/2020 – 1/11/2020</td></tr>
<tr><td>Final Report Due</td><td>1/11/2020</td></tr>
</table>
</td></tr>
<tr><td>Assumptions</td><td>

We will have access to all the resources required to complete this project.
Team members will not be changing during the project.
The scope of the project will not be changing.
Automation of the audit benchmark chart generation process will greatly increase the efficiency compared to the current process.
The resources and infrastructure required to run the new system will be readily available at ARPANSA or easy to acquire.
The database and software will be hosted internally at ARPANSA so security and privacy will not be a huge consideration.
The project team will be provided examples of the audit reports and Excel spreadsheet so that we can properly understand the current workflow and processes.
The client's will be providing information to the team and answering questions so that we can understand the data in the Excel spreadsheet

</td></tr>
</table>

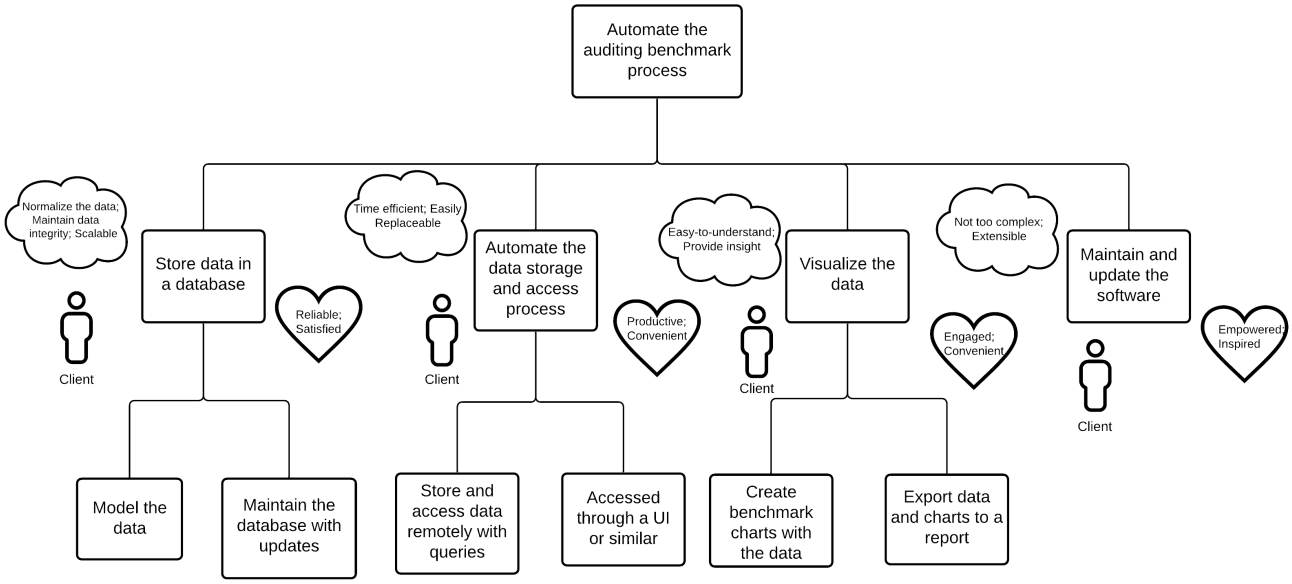
2.5 Risk Management

This page documents the potential risks associated with the build and delivery of this project.

	Risk	Probability	Impact	Action/Mitigation	Comments
1	Continuity	Medium	High	• Clear documentation to allow for efficient handover. • Logical completion of artifacts that are in their build stage.	• Client expectations are for a product that if not complete, can be handed over to the next group to continue.
2	External obstacles to work efficiency	High	Medium	• Remote working solutions: • Jira, slack, zoom, drive, etc. • Clear communication and sprint management. • Clear division of responsibilities.	• This includes the impacts of COVID19.
3	Client Maintainability	High	Medium	• Time will be dedicated during handover to allow for the training of the client to maintain and modify the software solution.	• The client lacks technical skills to maintain the software solution.
4	Internal obstacles to work efficiency	High	High	• Dedicate time to learning and development, whether that be during the semester or the semester break.	• This includes the lack of experience across the implemented technology stack.

2.6 Motivational Model

Who	Do	Be	Feel
Clients	Store data in database	• Normalize the data • Maintain data integrity • Scalable	• Reliable • Satisfied
Clients	Automate the data storage and access process	• Time efficient • Easily repeatable	• Productive • Convenient
Clients, Radiation Oncologist	Visualize the data	• Easy-to-understand • Provide insight	• Engaged • Convenient
Clients,	Maintain and update the software	• Not too complex • Extensible	• Empowered • Inspired



2.7 User Personas

User Personas

Name/Role	Details	Goals/Motivation	Frustrations
Medical Physicist	• A person working at ARPANSA who will be collecting the audit results from various medical oncology facilities. • They will be performing analysis on the audit results and creating charts and reports to document the data. • They will be the main user of this project.	• Make data storage aspect of the auditing process more efficient. • Make it easier to ease new physicists into the auditing process.	• Very busy with other work and dealing with the audit data is very time consuming. • Limited programming experience so they do not know how to approach the problem. • The Excel spreadsheet is getting too complicated and bloated
Radiation Oncologist	• The sites where the audits take place. • They receive the reports from their respective audits so they can compare the performance of their linear accelerators with other sites.	• Benchmark the performance of their machines with other facilities in the nation. • Receive accurate results from the audit.	• It may take some time to receive the results from the audit.
Future Contractor	• A person who will take over the project after completion in order to develop further functionality. • Could be another student team or a team of software developers.	• Extend the functionality of the software at the request of the clients. • Add new auditing measures to the data in the database. • Add a custom UI to the program.	• Have to understand the software that has already been developed. • Have very little contact with previous contractors.
Staff with Software Development Experience	• A person working at ARPANSA who will be maintaining the project and ensuring that it continues functioning as expected. • They have some software development experience so they can also extend upon the project with new functionality.	• Extend the functionality of the software. • Add new auditing measures to the data in the database. • Add a custom UI to the program.	• Have to understand the software that has already been developed. • Have very little contact with previous contractors.

2.8 User Stories

User Stories

ID	As a <Role>	I want to <do something>	So that I can <acheive a goal>	Priority
01	User (medical physicist)	pull input data from audits	analyze new data in conjunction with existing data in order to produce findings for reports.	Must have
02	User (medical physicist)	record and input new data garnered from an onsite audit	conduct analysis on input data to generate reports	Must have
03	User (medical physicist)	design new approaches to analyze the audit data	produce differing reports as required, on an ongoing basis.	Should have
04	User (medical physicist)	input data to a database in an automated fashion	focus on the analysis of the data rather than the process of recording it in a database.	Must have
05	User (medical physicist) /Radiation Oncologist	ensure data is secure at rest and in transit	protect sensitive information can be recorded and analyzed in an ethical and legal manner	Should have
06	User (medical physicist)	have a stable system	analyze the data in a convenient manner, when required (available)	Should have
07	Future contractor	extend upon the platform	scale the platform to run more complex analysis and accept new types of data as input	Should have
08	User (medical physicist)	access the shared external environment efficiently and from any machine	view the analysis generated by other medical physicists in the team, even when on the third-party client site.	Should have
09	User (medical physicist)	have data stored in a persistent storage	complete analysis on an ongoing basis using new and old data.	Must have
10	User (medical physicist)	create charts from audit data that can be exported	present it in the fashion desired by a third party, usually in a report.	Should have
11	User (staff with some development experience)	have the ability to create additional auditing templates	insert more detailed audits into the database in the future.	Could Have

2.9 Acceptance Criteria

User Story ID	User Story	Given	When	Then
1	As a user, I want to pull input data from audits so that I can analyze new data in conjunction with existing data in order to produce findings for reports.	I need to make an audit report for a facility	I check the data from the required audit and past audits	I put the data into a chart and create a report
2	As a user, I want to record and input new data garnered from an onsite audit so that I can conduct analysis on input data to generate reports.	I have just received the results for an audit	I obtain the data from a new audit	I store the data in a storage where it can be accessed later when required
3	As a user, I want to design new approaches to analyze the audit data so that I can produce differing reports as required, on an ongoing basis.	I obtain new ideas or approaches to analyzing the data	I analyze the data from all the audits	I select new queries for the data and analyze them in a report in a different way
4	As a user, I want to input data to a database in an automated fashion so that I can focus on the analysis of the data rather than the process of recording it in a database.	I have just received the results for an audit	I obtain the data from a new audit when I am busy with other tasks	I can easily store the data in a database in a quick and efficient manner
5	As a user, I want to ensure data is secure at rest and in transit so that I can protect sensitive information can be recorded and analyzed in an ethical and legal manner.	I have just received the results for an audit	I am storing the data	I can make sure that the data is secure and correct
6	As a user, I want to have a stable system so that I can analyze the data in a convenient manner, when required (available).	I need to make an audit report for a facility	I check the data from the required audit and past audits	I can access the data without worrying about the availability or the system crashing
7	As a future contractor , I want to extend upon the platform so that I can scale the platform to run more complex analysis and accept new types of data as input.	I am contracted to work on the system	I need to make changes to the system to update it	The system can accept new changes without anything else breaking in the process
8	As a user, I want to access the shared external environment efficiently and from any machine so that I can view the analysis generated by other medical physicists in the team, even when on the third-party client site.	I need to access the data or analysis	I am not using my own personal computer	The system can be accessed remotely and is not stored on a single machine
9	As a user, I want to have data stored in a persistent storage so that I can complete analysis on an ongoing basis using new and old data.	I need to make an audit report for a facility	I am retrieving data from the storage	I can access data that has been stored at any time period since the data storage started operating
10	As a user, I want to create charts from audit data that can be exported so that I can present it in the fashion desired by a third party, usually in a report.	I need to make an audit report for a facility	I am creating the charts for the report	I have control over the way the charts and reports are presented
11	As a staff with development experience, I want to have the ability to create additional auditing templates so that I can insert more detailed audits into the database in the future.	I have a new approach to measuring data from an audit	I need to make changes to the audit storage system	I can easily add new fields and store new audit data without breaking past audits

3 Architecture Design

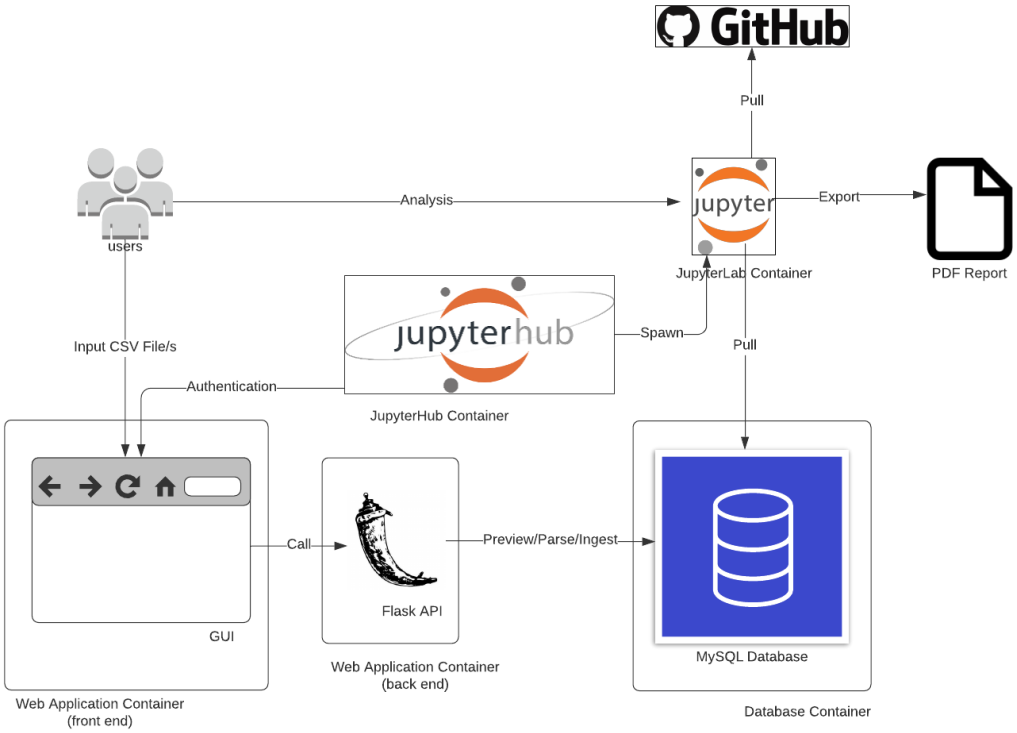
This section describes the architecture of our design and how the different components interact with each other. It also details the tools and libraries /frameworks we employed in developing this system.

Contents

- 2.1 Architecture Representation
- 2.2 Software Architecture

2.1 Architecture Representation

The diagram below shows the system architecture for our automated auditing system. The diagram shows the main subsystems and the interactions between them. Each subsystem is contained inside its own container which contains all the necessary packages and software to run.



Components

The following table describes the responsibilities of each module and the requirements they handle.

Component	Responsibilities	Requirements Handled (See 2.2 Functional Requirements and 2.3 Non-Functional Requirements)
Web Application Container (Front-End)	• Provide an interface where files and data can be uploaded to the database. • After uploading, the data can be ingested into the corresponding tables in the database. • Provide an interface to navigate to other aspects of the system.	• Workflow • System management • Licensing • Usability
Web Application Container (Back-End)	• Integrate the web application with the database. • Provides a REST API for interacting with the database. • Handles the data uploaded from the front-end and maintains the consistency in the database,	• Workflow • System management • Licensing
JupyterHub Container	• Provides authentication and security to the system. • Spawns instances of JupyterLab from contents contained inside a repository. • Maintains the consistency and version control for notebook instances.	• Security • Licensing

JupyterLab Container	• Provide an interface where data can be queried into dataframes and analysed. • Notebooks can generate charts and tables from the audit data which can be output into other formats. • Notebooks can be converted into other formats such as a pdf to make the audit reporting process more convenient.	• Analysis • Usability • System management • Licensing
Database Container	• Models the audit data in a relational model. • Store current and historical audit data that is used for data analysis. • Provide views for commonly joined tables that are useful for data analysis.	• Licensing • Reliability • Supportability

2.2 Software Architecture

Architectural Patterns

We did not make much use of architectural patterns in our code but we structured our front-end and back-end in a dumb and smart way.

Library/Frameworks

This section describes the main libraries/frameworks used by the system to provide the functionality the system will be based.

Library /Framework	Reason	Version	Environment
Python	• The clients are more familiar with Python code so it makes sense to choose Python with its extensive library for data science and analysis. • All of us are also familiar with Python and have used it before. • Python is under an open source license making it free to use and distribute.	3.7	• Development
mysql-connector-python	• A Python library that connects to a MySQL server. • The official option provided by MySQL to connect to a MySQL server using Python.	8.0.21	• Development
React.js	• We are more familiar with React and there would be more of a learning curve with other options. • We needed a library for building a front-end user interface. • It makes sense to just build a single page application for this system and React is built for that. • React is open source making it free to use and distribute.	16.13.1	• Development
Python Flask	• The clients are more familiar with Python code and this is a Python framework for web application back-end development. • We are implementing an API to standardize and automate the process for inserting data into the database. • Only option since we are using mainly Python.	1.1.2	• Development
Pandas	• One of the most widely used Python libraries in data science. • Represents data clearly in a 2d table-like manner and can handle large amounts of data. • Includes functions for analyzing data. • Can read/write data directly into/from a pandas dataframe using mysql-connector-python. • Another option would by numpy arrays but pandas is more suitable due to the data being tabular instead of multi-dimensional.	1.1.2	• Development
Docker	• We also needed a tool to manage our development and deployment platform and Docker Hub is a great way to develop an application and deploy it in one package. • We can run our system in containers with all the necessary libraries which makes it easier to manage compatibility issues. • Docker is also portable and can run on a variety of platforms. • Since we do not know the specifications of our client's servers, packaging our system in Docker containers will help with any compatibility issues. • Part of the CIS standard.	19.03	• Development • Production

Development Environment

The development environment and tools used to develop the system is described below.

Component	Tools	Reason
All	• GitHub	• A version control system is important for collaborative project work and GitHub allows us to manage our software in a repository. • We chose GitHub because we are more familiar with it. • More details of our repository structure can be found in section 5.
Web Application Container (Back-End)	• Postman	• Great for developing and testing an API. • Familiar with using Postman. • Can create tests for an API easily.

JupyterHub Container	• JupyterHub	• Provide security by adding in a token authentication service. • Integrates nicely with JupyterLab.
JupyterLab Container	• JupyterLab	• The clients are familiar with coding in Jupyter Notebooks and it is great for data analysis. • Jupyter provides an interface to run code and see the output quickly. • Can query a database and output the data in a chart using Python libraries. • Not as complex as creating an API and UI to do a similar task. • Can use an extension to spawn a repository from GitHub.
Database Container	• MySQL Workbench • MySQL Server	• We needed to develop a database to store historical audit data. • We are more familiar with MySQL workbench and it has a free version with sufficient functionality for this project. • MySQL workbench provides a convenient way to transform a database model into a script which can be run to create the database tables. • We are not working with complex queries or require fast read/write speeds so MySQL is a suitable platform. • Part of the CIS standard.

4 Use Cases

This section describes the users of our system and some example use cases for it. More information on how to accomplish each use case can be found in the tutorials in section 6.

Contents

- 4.1 Actors
- 4.2 Use Cases

4.1 Actors

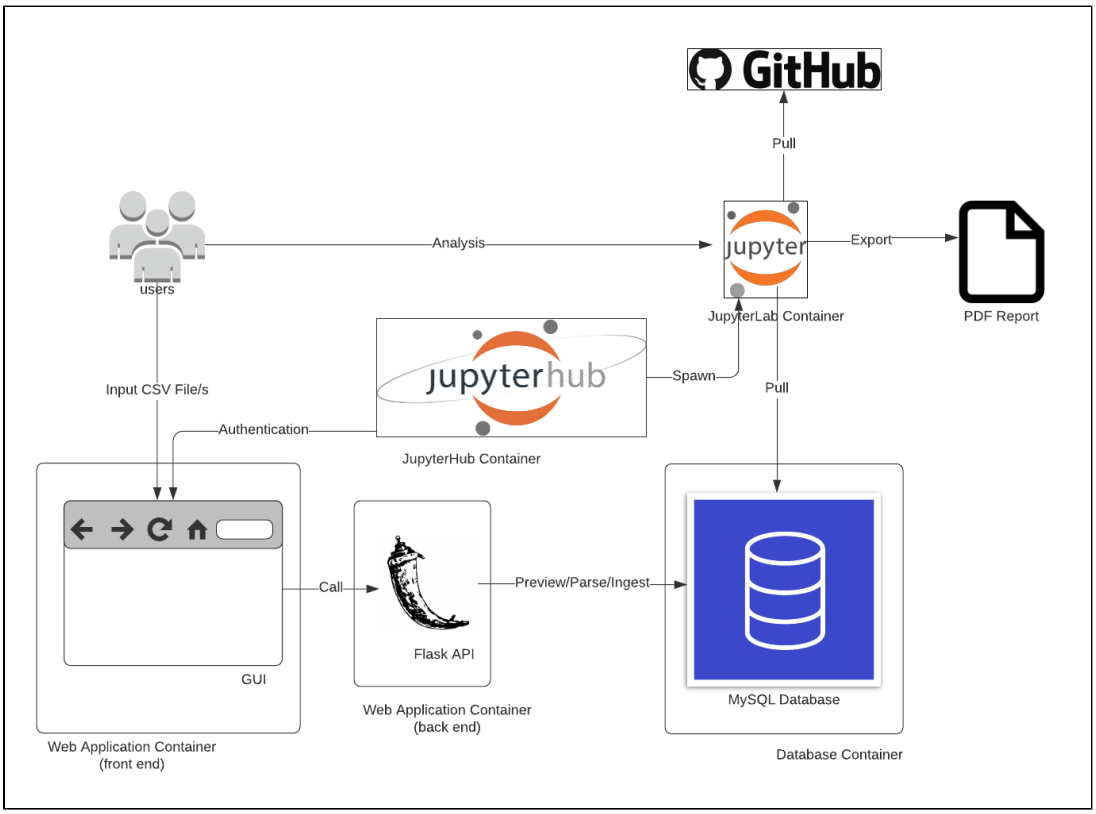
Actor	Description
Medical Physicist	• A person working at ARPANSA who will be collecting the audit results from various medical oncology facilities. • They will be performing analysis on the audit results and creating charts and reports to document the data. • They will be the main user of this project.

4.2 Use Cases

Use Case ID	Use Case Name	Pre-conditions	Main Events Flow	Alternative Flows	Exception Flows	Post-conditions	Additional Information
UC01	Install the containers for the system.	A medical physicist has pulled the contents from https://github.com/COMP90082-2020-SM2-AA-Wombat/platform and has DockerHub installed.	1. Navigate to the root directory in a terminal. 2. Navigate to the scripts directory. 3. Run startup.sh.	• [AF01] Running the shutdown.sh script instead will shutdown the system. • [AF02] Running the reset.sh script instead will reset the system. • [AF03] Running the delete_all.sh script instead will remove all the containers from the system.	• [EF01] The device running the startup.sh script may not have enough memory. • [EF02] When an exception occurs, the system will log the exception in the terminal.	• The system will be running in the background. • It can be navigated by opening a browser and navigating to ports 3000 for the front-end or 8000 for Jupyter Lab in your ip.	More information can be found in section 6.4.
UC02	Create an account	UC01	1. Go to http://<server ip>:8000 in a browser. 2. Click on the sign up button. 3. Create an account with the desired username and password.	• [AF04] If another account is created, it will need to be authorised by the first account by navigating to http://<server_name>:8000/hub/authorize. • [AF05] Using a username that already exists will not create an account.	• [EF02]	• An account should be created with the desired username and password. • The user will need to be authorised if it is not the first account created.	More information can be found in section 6.3.
UC03	Insert audit data into the database using a CSV file.	UC02	1. Go to http://<server ip>:3000 in a browser. 2. Login with their username and password. 3. Click on the upload files button. 4. Upload a CSV file with the name of the table you wish to upload into. 5. Click on the send button.	• [AF06] Clearing the uploaded file will cancel the process. • [AF07] Duplicate data will display a message informing that the data already exists in the database. • [AF08] Missing data fields will display a message informing that data fields are missing. • [AF09] Incorrect foreign keys will display a message informing that the foreign keys cannot be found.	• [EF02]	• The database table should be updated with every row in the CSV file.	More information can be found in section 6.1 and 6.2.
UC04	Insert audit data into the database using a dropdown.	UC02	1. Go to http://<server ip>:3000 in a browser. 2. Login with their username and password. 3. Click on the insertions button. 4. Choose a table in the dropdown. 5. Fill in the data. 6. Click on the submit button.	• [AF07], [AF08] and [AF09]	• [EF02]	• The database table should be updated with every row in the CSV file.	More information can be found in section 6.1 and 6.2.

UC05	Spawn a Jupyter Lab instance and clone a repository.	UC01 and pulled the contents from the repository https://github.com/COMP90082-2020-SM2-AA-Wombat/notebooks.	1. Go to http://<server ip>:8000 in a browser. 2. Login with their username and password. 3. Click on the Git icon. 4. Click on the clone a repository button. 5. Clone https://github.com/COMP90082-2020-SM2-AA-Wombat/notebooks.	• [AF10] Clicking on the shared folder will take you to a directory that is shared across other machines running this system. • [AF11] Cloning a different repository should also work.	• [EF02]	• The repository contents should appear in the sidebar.	More information can be found in section 6.3.
UC06	Retrieve audit data from the database.	UC03/UC04 and UC05	1. Open audit_queries.ipynb. 2. Run the notebook.	• [AF11] Opening a different notebook file should also work.	• [EF02] • [EF03] If an exception occurs in the notebook then it will be displayed in the notebook.	• The notebooks should retrieve the data and output it in tables or figures.	More information can be found in section 6.3.

5 Project Implementation



1. Platform

	Component	Type	Purpose	Documentation	Code Reference	Notes
1	Docker Compose	Docker Orchestration	Orchestrates the deployment of the service via Docker.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	
2	db	Docker Container	Hosts the MySQL database. It uses a standard MySQL image that is initialized by 3 separate initialization scripts.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts	SQL init scripts: 1. a_init_users.sql 2. b_init_objects.sql 3. c_init_dummy_data.sql These scripts will only run when the db data volume is empty (ie on the first startup of the service).
3	webapp-backend	Docker Container	Hosts the webapp's backend.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-backend	
4	webapp-frontend	Docker Container	Hosts the webapp's frontend.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend	
5	jupyterhub	Docker Container	Hosts jupyterhub, along with its proxy and PostgreSQL database.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/jupyterhub	
6	custom-lab	Docker Image	The image that is deployed via jupyterhub when a new user logs in.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/jupyterhub/custom-lab	
7	frontend_network	Docker Network	Network for the webapp frontend.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	

8	backend_network	Docker Network	Network for the webapp backend and MySQL database.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	
9	jupyterhub_network	Docker Network	Network for the JupyterHub container and its spawned containers.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	
10	db_data	Docker Volume	The volume that caters for persistence by the MySQL database.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	
11	jupyterhub_data	Docker Volume	The volume that caters to config/state persistence by the JupyterHub container.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/docker-compose.yaml	
12	scripts	Bash Scripts	Startup, shutdown, reset, and delete-all bash scripts.	N/A	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/scripts	• NOTE: delete_all.sh will affect all docker applications on the host. • delete_all.sh: deletes all volumes, containers, and networks on the host machine. • reset.sh: shuts down the application and starts it up again with no data loss. • shutdown.sh: shuts down the application. • startup.sh.sh: initializes the service by building all containers and running them.

2. Web Application - Front End

	Component	Type	Purpose	Documentation	Code Reference	Notes
1	Login	React.js /Javascript	Allows users to login to the web app (using Juptyer credentials.	5.2 Front-End Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend/src/pages/LoginPage	
2	CSV/JSON ingestion	React.js /Javascript	Allows users to add in csv and json files that are used to upload audits.	5.2 Front-End Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend/src/pages/CSVPage	
3	Dynamic Field ingestion	React.js /Javascript	Allows users to add a row into the database for any table. Loads the tables and fields to add dynamically.	5.2 Front-End Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend/src/pages/InsertionPage	
4	Profile	React.js /Javascript	Allows a user to view their profile.	5.2 Front-End Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend/src/components/Profile	
5	JupyterHub	React.js /Javascript	Allows users to navigate to directly to JupyterHub.	5.2 Front-End Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/webapp-frontend/src/components/NavBar	

3. Web Application - Back End

	Component	Type	Purpose	Documentation	Code Reference	Notes
1	Ingestion	Flask /Python	Provides the APIs to allow users to ingest data to be stored in the database	5.3 Back-End Implementation		
2	Authentication	Flask /Python	Provides the API to allow users to login using their juptyer credentials	5.3 Back-End Implementation		
3						
4						
5						
6						
7						
8						

9						
10						

4. Database

	Component	Type	Purpose	Documentation	Code Reference	Notes
1	aa_audit	MySQL Database Schema	Database storage for audit data	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/database/blob/master/aa_ddl_v5.sql	Standalone database (duplicated in below init scripts) For detailed data dictionary see documentation in 6.5
2	aa_audit_dummy_data_v6.sql	SQL Data Dump	Dummy Data	Used in 6 Tutorials	https://github.com/COMP90082-2020-SM2-AA-Wombat/database/blob/master/aa_dummy_data_v6.sql	Dummy Data for demo purposes (duplicated in below init scripts)
3	platform/mysql-db/**sql-scripts**/a_init_users.sql	Platform init script with SQL User Create Statements	Define Users and Roles to be used by Platform	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts/a_init_users.sql	Run when db container starts up F
4	platform/mysql-db/**sql-scripts**/b_init_objects.sql	MySQL Database Schema	Define Database Schema definition used by Platform	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/mysql-db/sql-scripts/b_init_objects.sql	Run when db container starts up For detailed data dictionary see documentation in 6.5
5	platform/mysql-db/**sql-scripts**/c_init_dummy_data.sql	SQL Data Dump	Dummy Data	Used in 7 Tutorials	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts	Run when db container starts up
6	'webapp_user'	Database User	A Writer role DB user for use by the data ingestion web app	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts/a_init_users.sql	Used by Front and Back end web applications to write to db
7	'jupyter_user'	Database User	A Reader role DB user for use by the JupyterHub notebooks	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts/a_init_users.sql	Used by JupyterHub to read data from DB
8	'dbmaintainer_user'	Database User	An Admin role DB user for user for performing maintenance on the database	5.5 Database Implementation	https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/mysql-db/sql-scripts/a_init_users.sql	Admin user - Not used directly by any application.

5. JupyterHub

	Component	Type	Purpose	Document Reference	Code Reference	Notes
1	jupyterhub	Docker Container	This is the container that hosts jupyterhub, and manages the deployment of containers (eg. custom-lab) when users log in.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/jupyterhub/jupyterhub_config.py	This docker image relies on jupyterhub_config.py to run.
2	custom-lab	Docker Container	This is the custom docker container spawned by JupyterHub when a new user logs in.		https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/jupyterhub/custom-lab	Docker image is built and stored on the host before the startup of the application.
3	jupyterlab-git	Jupyter Lab Plugin	This is a plugin for the custom-lab environment that allows users to pull/push from/to a remote git repository in github/gitlab /Bitbucket, as well as allowing version control capabilities using git in the local Lab environment.		https://github.com/jupyterlab/jupyterlab-git	This is installed and initialized via the custom-lab docker image.
4	jupyter-nativeauthenticator	Jupyter Hub Authenticator	A library that allows for simple user authentication and admin tools for the authorizing of users.		https://github.com/jupyterhub/nativeauthenticator	This package leverages a postgres db to store user credentials.

5.1 Platform Implementation

Overview:

The platform for this service consists of 4 permanent Docker containers, all deployed using the orchestration tool, Docker Compose. Docker is a PaaS (Platform as a Service) offering that utilizes OS-level virtualization to provide software in packages known as containers. Containers are naturally isolated from one another but can be harmonized using orchestration tools such as Docker Compose.

4 Permanent Containers:

1. db
2. webapp-backend
3. webapp-frontend
4. jupyterhub

Implementation

1. **db**
    - Base image: mysql:8.0.22
    - Volumes:
        - /mysql-db/sql-scripts:/docker-entrypoint-initdb.d/:ro for init sql scripts
        - db_data:/var/lib/mysql for persistence of db data
    - Networks:
        - backend_network
        - jupyterhub_network
    - Config/Init Scripts:
        - a_init_users.sql
        - b_init_objects.sql
        - c_init_dummy_data.sql
2. **webapp-backend:**
    - Base image: python:3.6
    - Volumes: ./webapp-backend/app:/app
    - Ports: 5000:5000
    - Networks:
        - frontend_network
        - backend_network
        - jupyterhub_network
    - Config/Init Scripts:
        - FLASK_APP=flaskr FLASK_ENV=development flask run
        - settings.py
3. **webapp-frontend**
    - Base image:
        - Build: node:13.12.0-alpine
        - Production: nginx:stable-alpine
    - Ports: 3000:80
    - Networks:
        - frontend_network
    - Config/Init Scripts:
        - package.json
        - .env
4. **jupyterhub**
    - Base image: jupyterhub/jupyterhub 1.2
    - Volumes:
        - /var/run/docker.sock:/var/run/docker.sock jupyterhub init and configs
        - jupyterhub_data:/srv/jupyterhub jupyterhub data/state persietence
    - Ports: 8000:8000
    - Networks: jupyterhub_network
    - Config/Init Scripts:
        - jupyterhub_config.py

References:

- https://docs.docker.com/compose/
- https://hub.docker.com/

5.2 Front-End Implementation

Overview

- React.js
- Node

How to run

Locally

To run the frontend locally, you will need: Node

Steps:

1. npm install
2. npm run start

Docker compose

Refer to the docker compose documentation on how to run the system as a whole.

Feature set

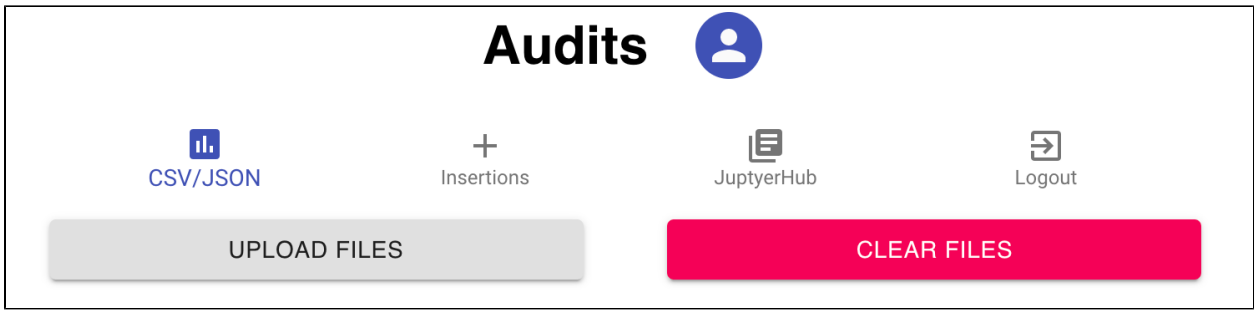
Login Page

Logging in is federated throughout our entire application. More specifically, users must login with their JuptyerHub credentials.

Login page

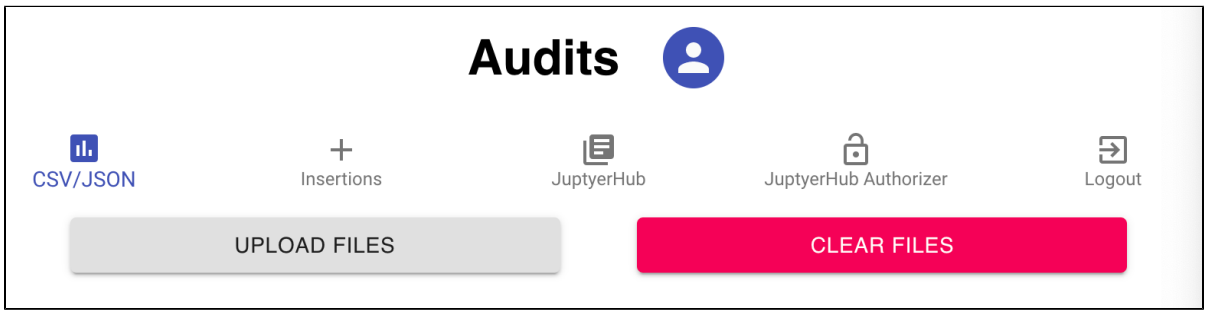
Navigation Bar

Normal Users



Users are able to navigate to the CSV/JSON page, the Insertion Pages, to JuptyerHub and finally to log out.

Admins



Admins have access to the same navigation as normal users, but will also have access to authorizing Jupyter users.

CSV/JSON Page

Purpose

This page is the default page the user will be guided to upon logging in. This page is designed to allow for inserting audits. It has the capability to add multiple audits at a time.

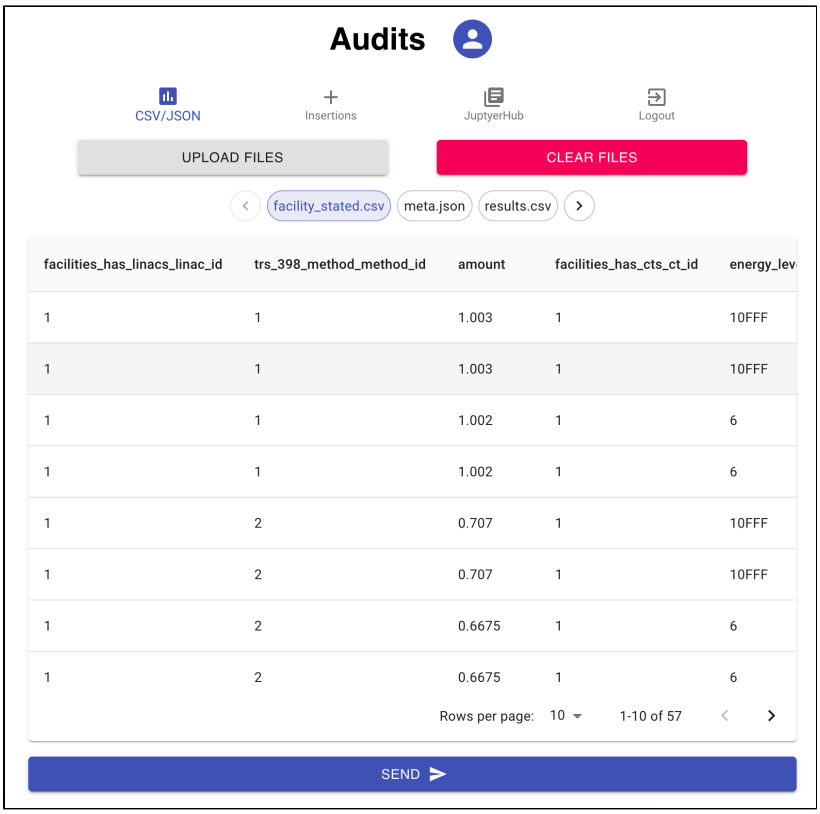
Definition

An Audit is defined by three files:

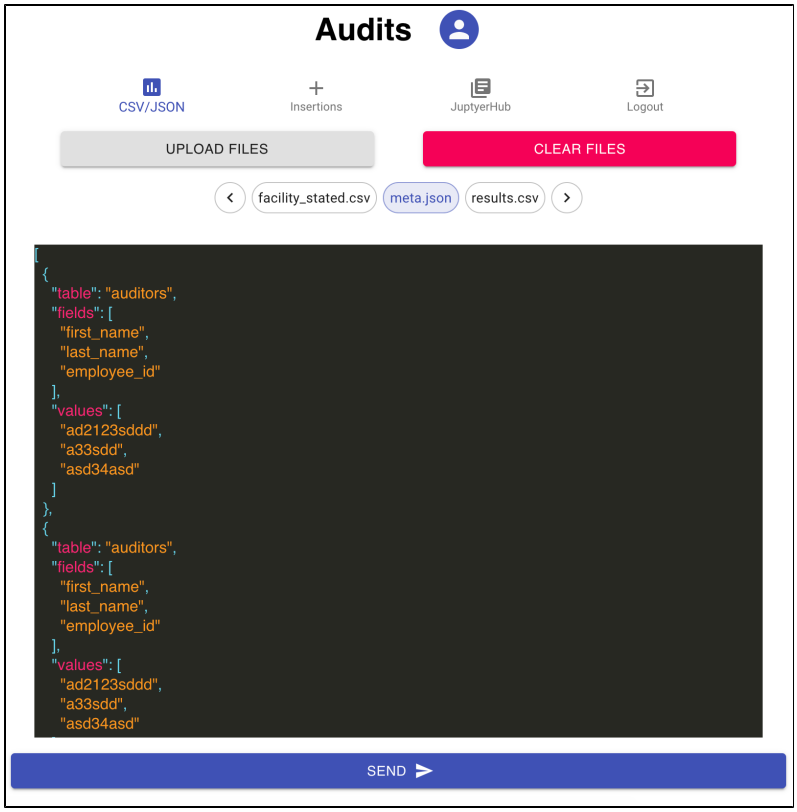
- A csv which has "results" in its name (ie. "results.csv" or "hello_results_world.csv"). This file will contain information about the results of an audit.
  - example format: results.csv
- A csv which has "facility_stated" in its name (ie. "results.csv" or "hello_facility_stated_world.csv"). This file will contain information about the facilities states of an audit.
  - example format: facility_stated.csv
- A json file which contains all the necessary meta data to complete an audit (e.g the auditors). This is information which may already be stored. For example, in the instance of adding auditors, if the auditors are already in the database, there is no need to add this meta data.
  - example format: meta.json
  - NOTE: readers will notice that the meta.json file is a list. This is because order matters when it comes to insertions in the database.

Inserting multiple files

Multiple files can be uploaded at once to be inserted. An interface is also provided to read through each of the csv/json files. This allows a user to double check all their values before uploading the files.



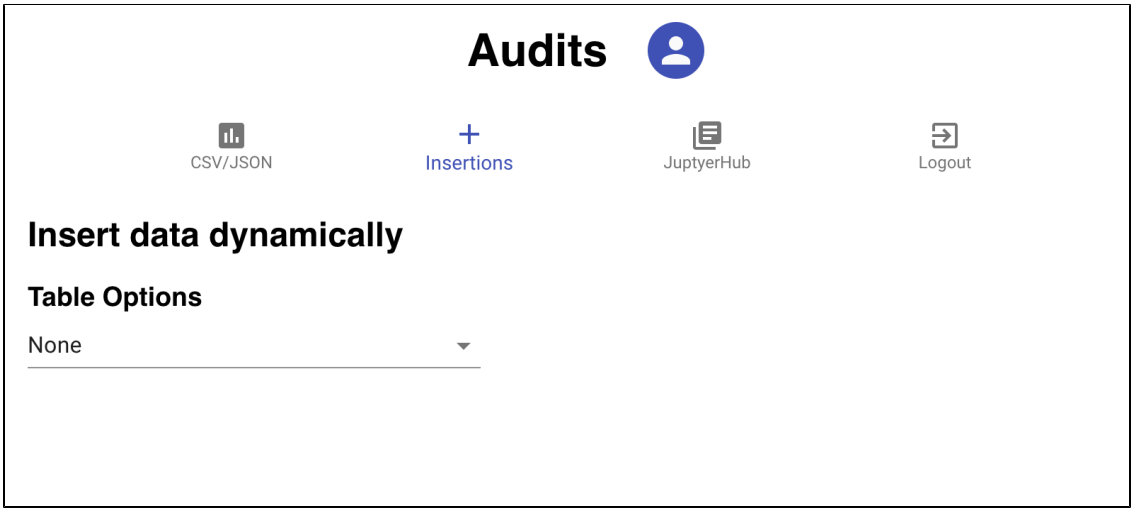
CSV representation



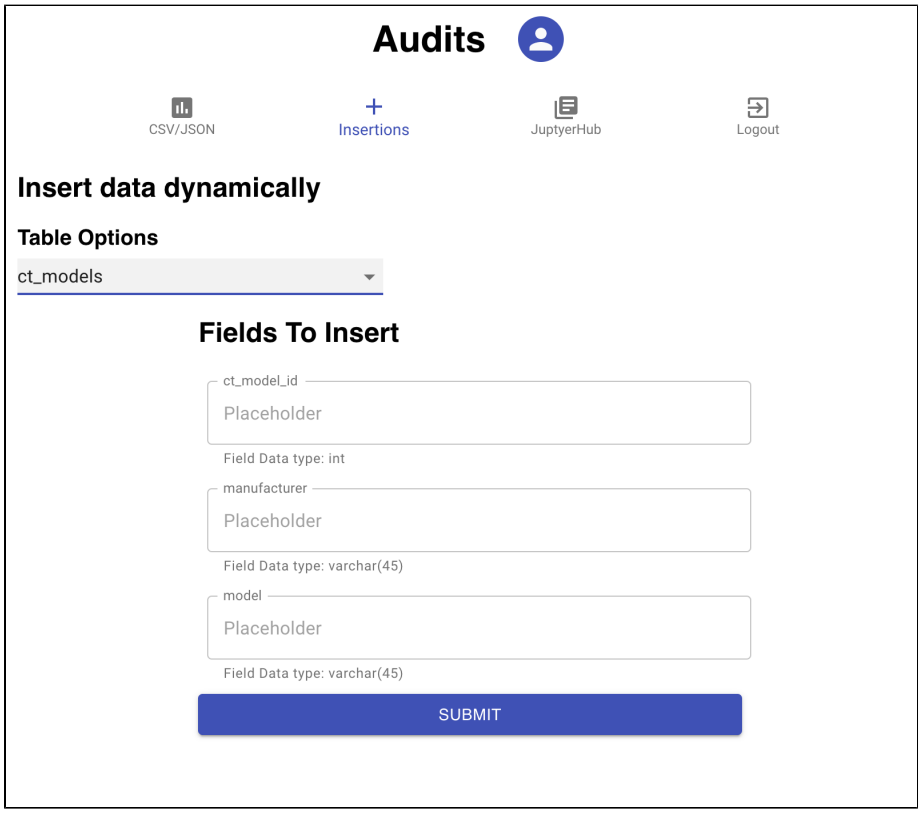
JSON representation

Dynamic Field Insertions Page

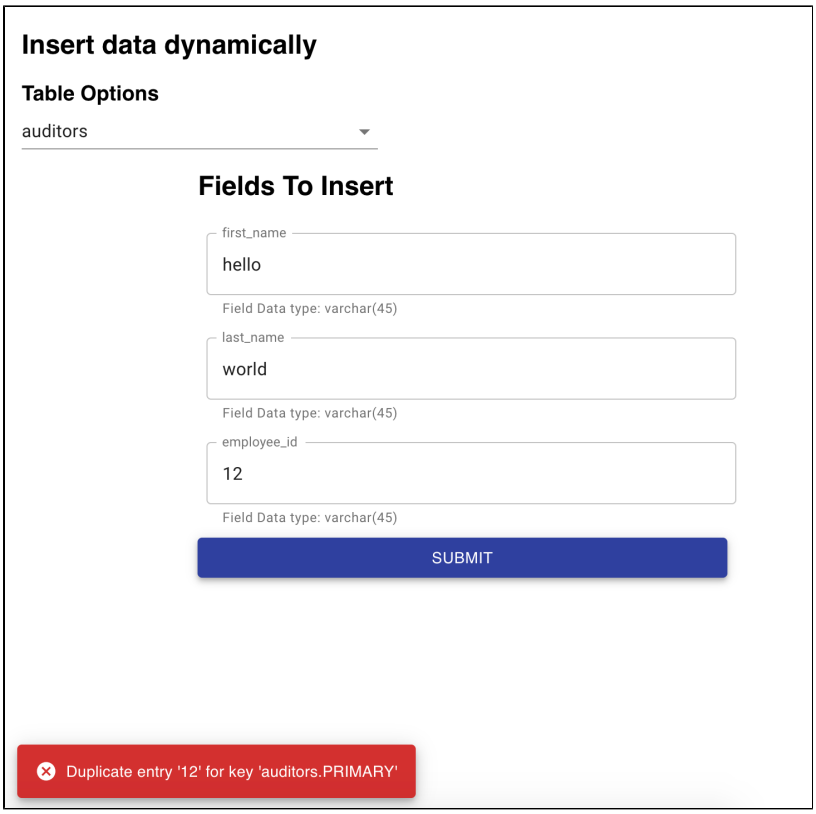
This page is designed to allow for dynamic addition of fields into the database. Users will be able to see all the tables present in the database, and be allowed to insert a row in that database provided of course that they maintain the correct database integrity. Errors will be displayed to help the user if they fail in an attempt to insert a field.



Dropdown to access all available tables in the database



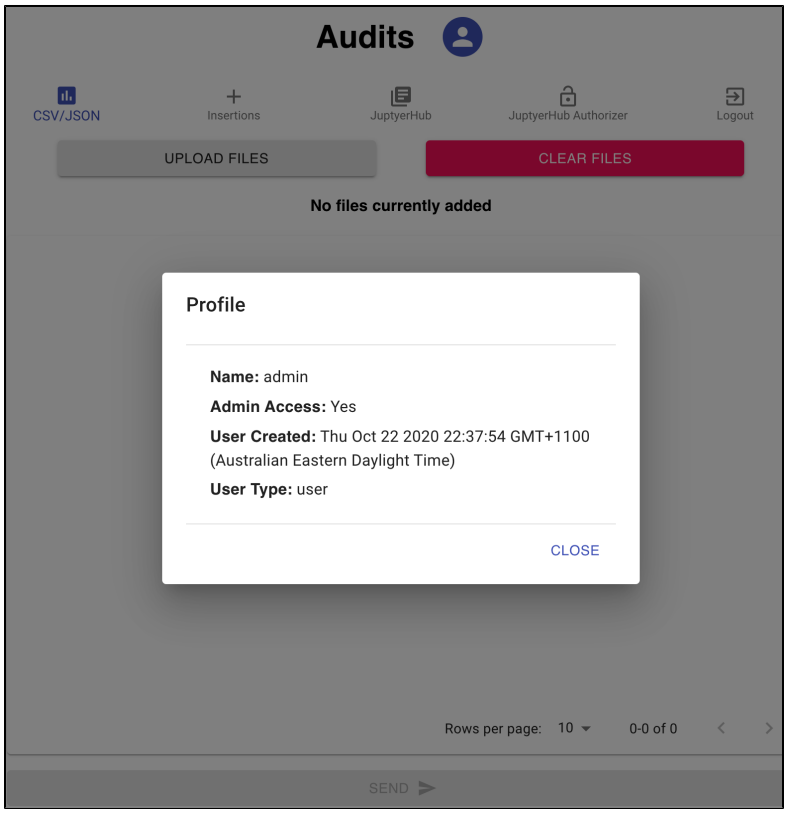
Dynamic tables show fields to insert



Error checking displayed

Profile

Users can see their own profile as is shown below:



Overview

- React.js
- Node

How to run

Locally

To run the frontend locally, you will need: Node

Steps:

1. npm install
2. npm run start

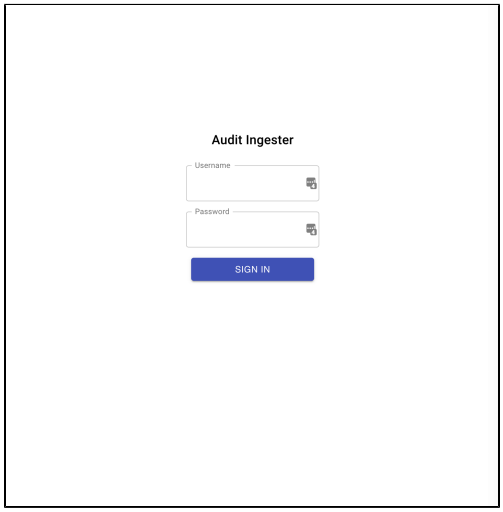
Docker compose

Refer to the docker compose documentation on how to run the system as a whole.

Feature set

Login Page

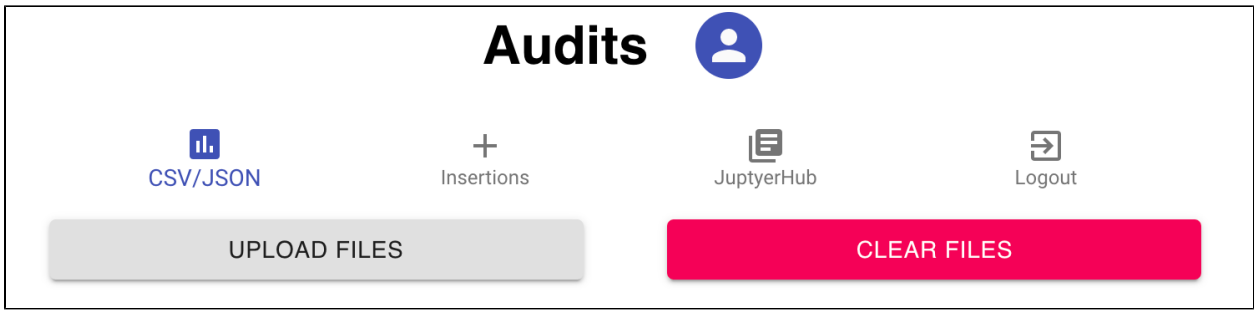
Logging in is federated throughout our entire application. More specifically, users must login with their JuptyerHub credentials.



Login page

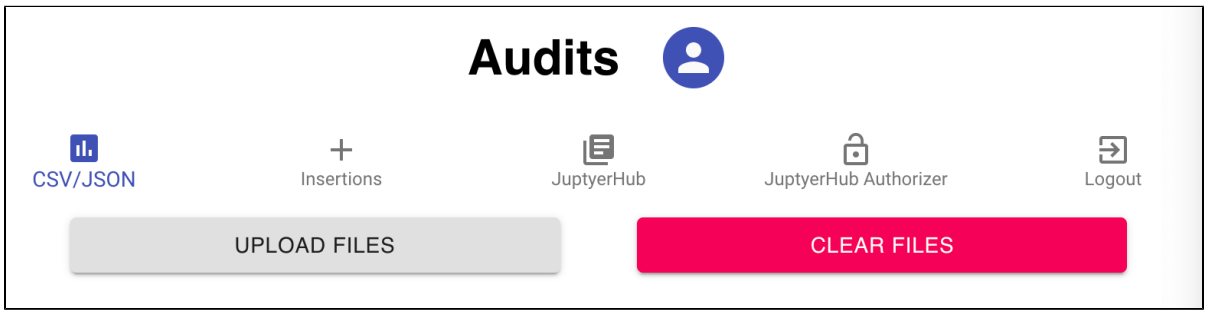
Navigation Bar

Normal Users



Users are able to navigate to the CSV/JSON page, the Insertion Pages, to JuptyerHub and finally to log out.

Admins



Admins have access to the same navigation as normal users, but will also have access to authorizing Jupyter users.

CSV/JSON Page

Purpose

This page is the default page the user will be guided to upon logging in. This page is designed to allow for inserting audits. It has the capability to add multiple audits at a time.

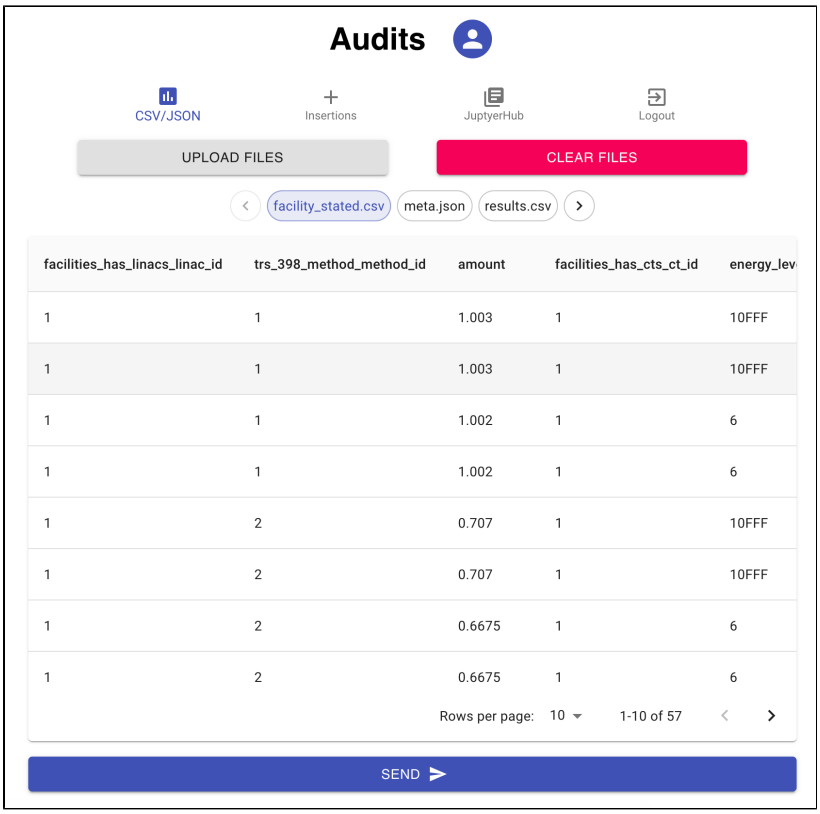
Definition

An Audit is defined by three files:

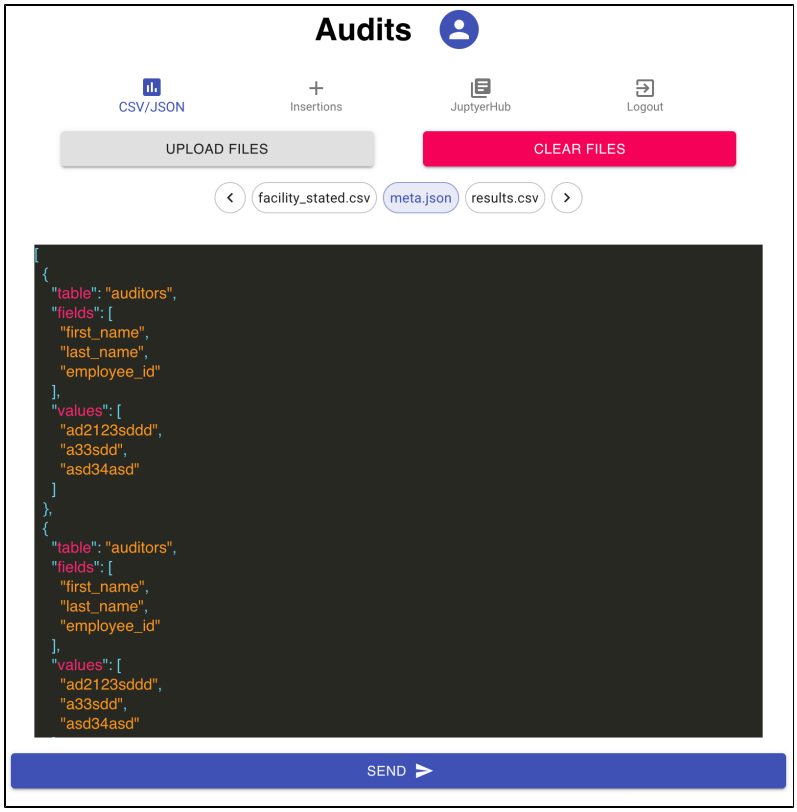
- A csv which has "results" in its name (ie. "results.csv" or "hello_results_world.csv"). This file will contain information about the results of an audit.
  - example format: results.csv
- A csv which has "facility_stated" in its name (ie. "results.csv" or "hello_facility_stated_world.csv"). This file will contain information about the facilities states of an audit.
  - example format: facility_stated.csv
- A json file which contains all the necessary meta data to complete an audit (e.g the auditors). This is information which may already be stored. For example, in the instance of adding auditors, if the auditors are already in the database, there is no need to add this meta data.
  - example format: meta.json
  - NOTE: readers will notice that the meta.json file is a list. This is because order matters when it comes to insertions in the database.

Inserting multiple files

Multiple files can be uploaded at once to be inserted. An interface is also provided to read through each of the csv/json files. This allows a user to double check all their values before uploading the files.



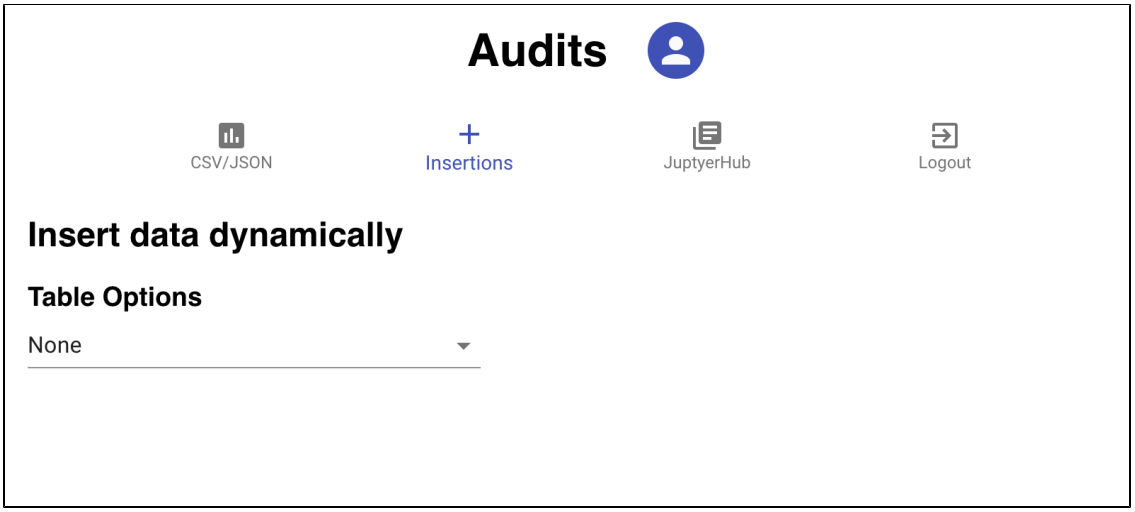
CSV representation



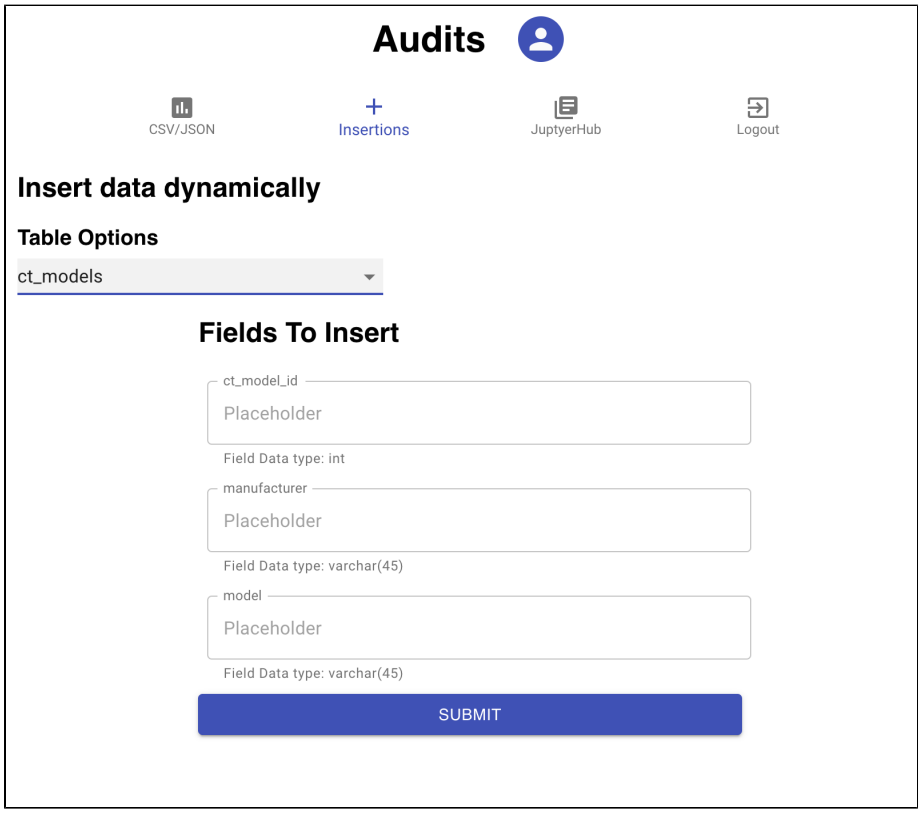
JSON representation

Dynamic Field Insertions Page

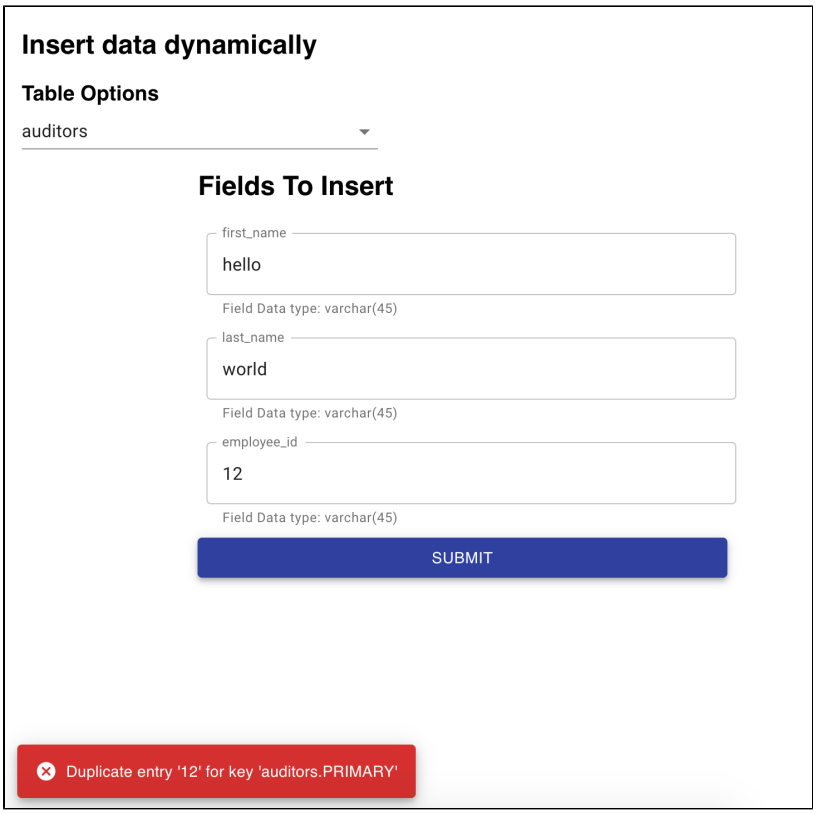
This page is designed to allow for dynamic addition of fields into the database. Users will be able to see all the tables present in the database, and be allowed to insert a row in that database provided of course that they maintain the correct database integrity. Errors will be displayed to help the user if they fail in an attempt to insert a field.



Dropdown to access all available tables in the database



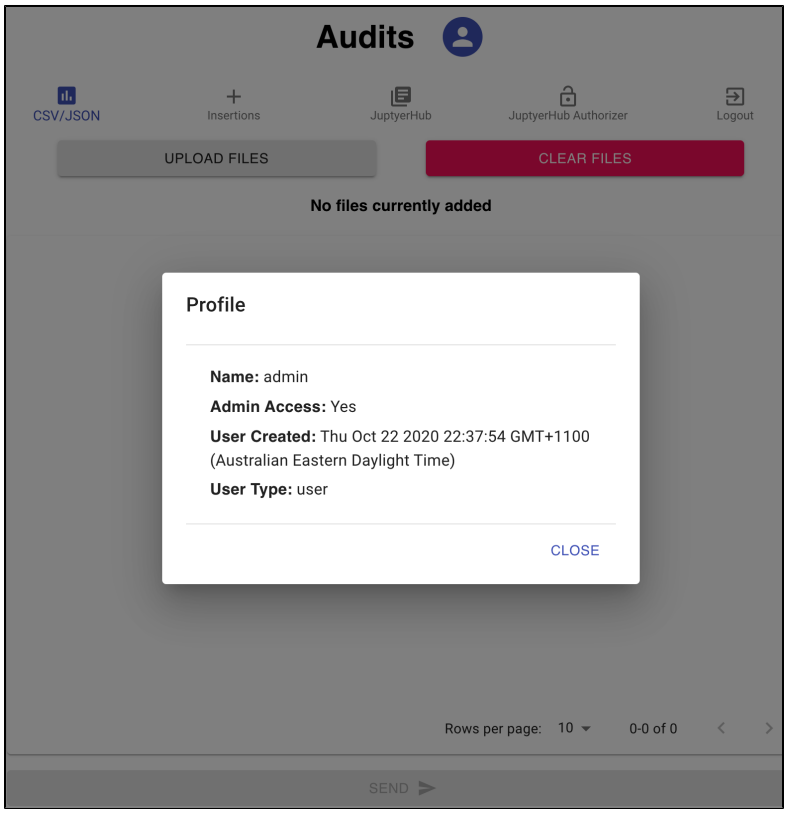
Dynamic tables show fields to insert



Error checking displayed

Profile

Users can see their own profile as is shown below:



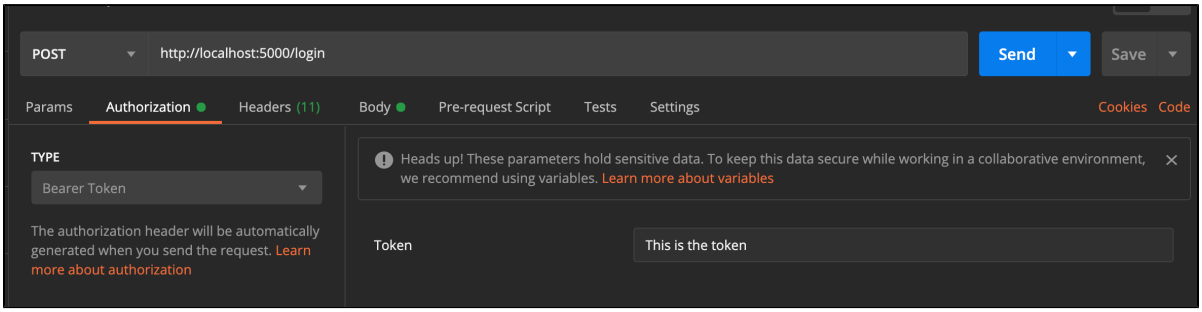
5.3 Back-End Implementation

Overview

The backend is completed in Flask, and is a REST API.

Endpoints

Authorising an endpoint which is protected will need to add it like so:

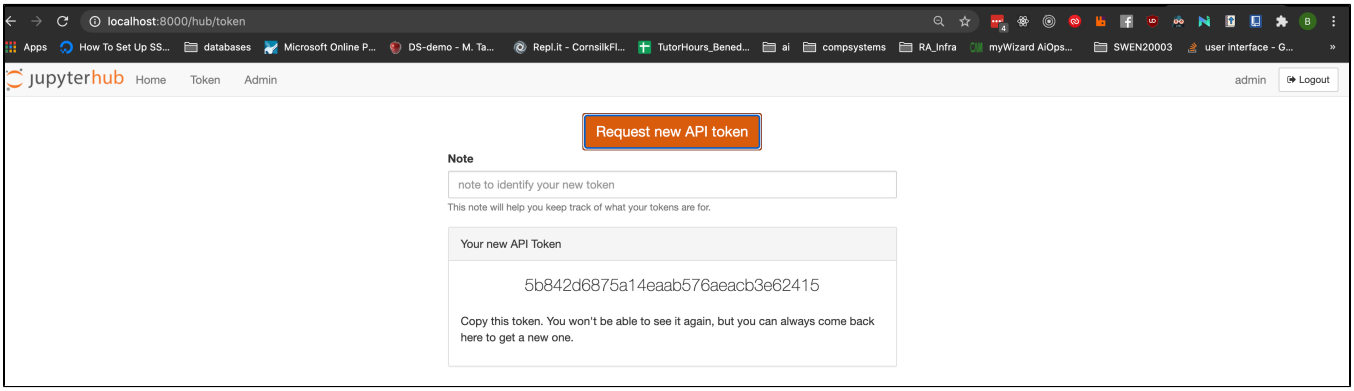


Request is being made via Postman

or alternatively by adding to the header of the request:

Authorization: Bearer THIS_IS_THE_TOKEN

Users can obtain a Jupyter Access token by going to the JuptyerHub Page under the /hub/token url, and refreshing to get a token:



In this case "5b842d6875a14eaab576aeacb3e62415" is the access token.

GET – /login

Description

This endpoint will allow a user to login.

Request Body

Login Request Body

```
{
    "username": "admin",
    "password": "admin"
}
```

Response Body

Login Success Response Body 200


```
{
    "token": "a6069ca6f01d4acca101f2b483abe6c1",
    "user": {
        "admin": true,
        "created": "2020-10-19T01:14:53.827172Z",
        "groups": [],
        "kind": "user",
        "last_activity": "2020-10-21T04:52:16.152501Z",
        "name": "admin",
        "pending": null,
        "server": null,
        "servers": null
    },
    "warning": "Using deprecated token creation endpoint /hub/api/authorizations/token. Use /hub/api/users/:user
/tokens instead."
}
```


Login Fail Response Body 401


```
{
   "message": "Access Forbidden"
}
```


GET – /table-fields

Description

This endpoint will allow users to get all table fields.

NOTE: This is a protected route. As such requests to these endpoints must include in the Authorization Header a Jupyter Access Token.

Response Body

Table fields Success 200 Response Body


```
[
        "table1": [
        {
            "field": "id",
            "type": "int"
        }
        ],
        "table2": [
        {
            "field": "first_name",
            "type": "varchar(45)"
        },
                {
            "field": "last_name",
            "type": "varchar(45)"
        },
        ],

]
```


POST – /bulk-fields

Description

This will allow users to add multiple rows to the database. It is important to note that the request body is specifically a list since order of insertions to the database is significant.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Bulks fields Request Body

```
[{
    "fields": ["method_id", "description"],
    "values": ["10", "1"],
    "table": "trs_398_method"
}]
```

Response Body

Bulks fields 200 Response Body

```
{
        "message": "Successfully added bulk data"
}
```

POST – /fields

Description

This will allow users to add a row to the database.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Fields Request Body

```
{
        "table": "auditors",
        "fields": ["first_name", "last_name", "employee_id"],
        "values": ["Johny", "Boy", "as123Sd3"]
}
```

Response Body

Fields 200 Response Body

```
{
        "message": "Successfully added field data"
}
```

POST – /bulk-tables

Description

This will allow users multiple rows to multiple tables in a bulk ordered manner

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Bulks Table Request Body

```
[
        {
                "table": "trs_398_method",
                "insertions":[
                        {
                                "fields": ["method_id", "description"],
                                "values": ["654", "1"]
                        },
                        {
                                "fields": ["method_id", "description"],
                                "values": ["53", "1"]
                        }
                ]
        },
        {
                "table": "phantoms",
                "insertions":[
                        {
                                "fields": ["phantom_id", "name"],
                                "values": ["654", "1"]
                        },
                        {
                                "fields": ["phantom_id", "name"],
                                "values": ["53", "1"]
                        }
                ]
        }
]
```

Response Body

Bulk Table 200 Response Body

```
{
        "message": "Successfully added bulk data"
}
```

POST – /csv

Description

This will allow users to add a row to the database.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Request Files:

- A csv which has "results" in its name (ie. "results.csv" or "hello_results_world.csv"). This file will contain information about the results of an audit.
  - example format: results.csv
- A csv which has "facility_stated" in its name (ie. "results.csv" or "hello_facility_stated_world.csv"). This file will contain information about the facilities states of an audit.
  - example format: facility_stated.csv
- A json file which contains all the necessary meta data to complete an audit (e.g the auditors). This is information which may already be stored. For example, in the instance of adding auditors, if the auditors are already in the database, there is no need to add this meta data.
  - example format: meta.json
  - NOTE: readers will notice that the meta.json file is a list. This is because order matters when it comes to insertions in the database.

Response Body

CSV 200 Response Body

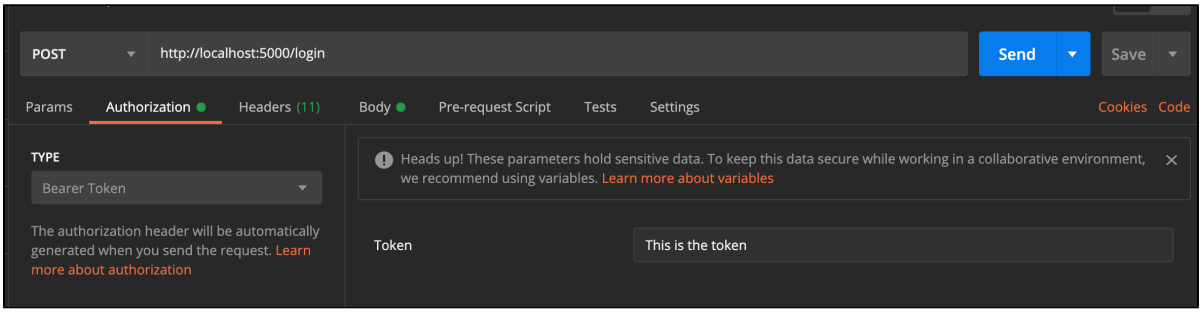
```
{
        "message": "Successfully added data"
}
```

Overview

The backend is completed in Flask, and is a REST API.

Endpoints

Authorising an endpoint which is protected will need to add it like so:

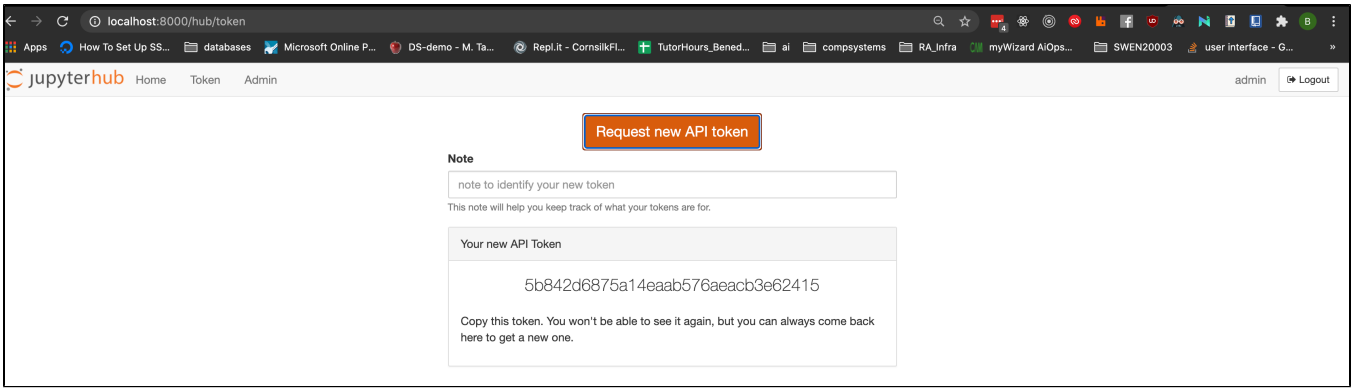


Request is being made via Postman

or alternatively by adding to the header of the request:

Authorization: Bearer THIS_IS_THE_TOKEN

Users can obtain a Jupyter Access token by going to the JuptyerHub Page under the /hub/token url, and refreshing to get a token:



In this case "5b842d6875a14eaab576aeacb3e62415" is the access token.

GET – /login

Description

This endpoint will allow a user to login.

Request Body

Login Request Body

```
{
    "username": "admin",
    "password": "admin"
}
```

Response Body

Login Success Response Body 200

```
{
    "token": "a6069ca6f01d4acca101f2b483abe6c1",
    "user": {
        "admin": true,
        "created": "2020-10-19T01:14:53.827172Z",
        "groups": [],
        "kind": "user",
        "last_activity": "2020-10-21T04:52:16.152501Z",
        "name": "admin",
        "pending": null,
        "server": null,
        "servers": null
    },
    "warning": "Using deprecated token creation endpoint /hub/api/authorizations/token. Use /hub/api/users/:user
/tokens instead."
}
```

Login Fail Response Body 401

```
{
   "message": "Access Forbidden"
}
```

GET – /table-fields

Description

This endpoint will allow users to get all table fields.

NOTE: This is a protected route. As such requests to these endpoints must include in the Authorization Header a Jupyter Access Token.

Response Body

Table fields Success 200 Response Body

```
[
        "table1": [
        {
            "field": "id",
            "type": "int"
        }
        ],
        "table2": [
        {
            "field": "first_name",
            "type": "varchar(45)"
        },
                {
            "field": "last_name",
            "type": "varchar(45)"
        },
        ],

]
```

POST – /bulk-fields

Description

This will allow users to add multiple rows to the database. It is important to note that the request body is specifically a list since order of insertions to the database is significant.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Bulks fields Request Body

```
[{
    "fields": ["method_id", "description"],
    "values": ["10", "1"],
    "table": "trs_398_method"
}]
```

Response Body

Bulks fields 200 Response Body

```
{
        "message": "Successfully added bulk data"
}
```

POST – /fields

Description

This will allow users to add a row to the database.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Fields Request Body

```
{
        "table": "auditors",
        "fields": ["first_name", "last_name", "employee_id"],
        "values": ["Johny", "Boy", "as123Sd3"]
}
```

Response Body

Fields 200 Response Body

```
{
        "message": "Successfully added field data"
}
```

POST – /bulk-tables

Description

This will allow users multiple rows to multiple tables in a bulk ordered manner

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Query Parameters

updateOrReplace:

- Can be "true" or "false" (i.e http://localhost:5000/bulk-tables?updateOrReplace=true
- Having this flag being true will allow for data to be updated if it already exists, and created otherwise.

Request Body

Bulks Table Request Body

```
[
        {
                "table": "trs_398_method",
                "insertions":[
                        {
                                "fields": ["method_id", "description"],
                                "values": ["654", "1"]
                        },
                        {
                                "fields": ["method_id", "description"],
                                "values": ["53", "1"]
                        }
                ]
        },
        {
                "table": "phantoms",
                "insertions":[
                        {
                                "fields": ["phantom_id", "name"],
                                "values": ["654", "1"]
                        },
                        {
                                "fields": ["phantom_id", "name"],
                                "values": ["53", "1"]
                        }
                ]
        }
]
```

Response Body

Bulk Table 200 Response Body

```
{
        "message": "Successfully added bulk data"
}
```

POST – /csv

Description

This will allow users to add a row to the database.

NOTE: This is a protected route. As such requests to this endpoint must include in the Authorization Header a Jupyter Access Token.

Request Files:

- A csv which has "results" in its name (ie. "results.csv" or "hello_results_world.csv"). This file will contain information about the results of an audit.
  - example format: results.csv
- A csv which has "facility_stated" in its name (ie. "results.csv" or "hello_facility_stated_world.csv"). This file will contain information about the facilities states of an audit.
  - example format: facility_stated.csv
- A json file which contains all the necessary meta data to complete an audit (e.g the auditors). This is information which may already be stored. For example, in the instance of adding auditors, if the auditors are already in the database, there is no need to add this meta data.
  - example format: meta.json
  - NOTE: readers will notice that the meta.json file is a list. This is because order matters when it comes to insertions in the database.

Response Body

CSV 200 Response Body


```
{
        "message": "Successfully added data"
}
```

5.4 Jupyter Implementation

Overview

- Jupyter is the technology of choice for this service, to allow for the analysis of data collected from ongoing audits by ARPANSA.
- Project Jupyter is the open-source initiative that produced the web based development environment, JupyterLab, as well as the associated collaboration technology, JupyterHub.
- This project utilizes both these technologies in creating a collaborative data analysis environment for the production of ARPANSA audit reports.
- At a high level, **JupyterHub** is the technology that allows multiple users to collaborate in a given organization while using **JupyterLab** for data analysis.

Implementation

1 - JupyterHub

- Code reference: https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/jupyterhub
- The below table describes some of the core configurations set in the jupyterhub_config.py.

Attribute	Explanation	Reference
DockerSpawner	Spawner Class	https://github.com/jupyterhub/dockerspawner
NativeAuthenticator	Authentication	https://native-authenticator.readthedocs.io/en/latest/quickstart.html
Networking	• hub_ip: 0.0.0.0 • hub connect ip: jupyterhub (host container)	https://jupyterhub.readthedocs.io/en/stable/getting-started/networking-basics.html

- The below table describes some of the core configurations for DockerSpawner set in the jupyterhub_config.py. These determine the features of the JupyterLab container that is deployed for the purpose of data analysis by end-users.

Attribute	Explanation	Reference
DockerSpawner	Spawner Class	https://github.com/jupyterhub/dockerspawner
jupyter/scipy-notebook	Spawner Default Image	https://hub.docker.com/r/jupyter/scipy-notebook
Networking	Spawner network name: jupyterhub_network	https://jupyterhub.readthedocs.io/en/stable/getting-started/networking-basics.html
jupyterhub-user-{username} juypterhub-shared-data	Volume (user data and shared data)	https://github.com/jupyterhub/dockerspawner/issues/239
default_url = '/lab'	JupyterLab used by default	https://jupyterlab.readthedocs.io/en/stable/

2 - JupyterLab

- The docker container set to deploy upon user login is hardcoded to be 'custom-lab' (https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/tree/master/jupyterhub/custom-lab)
- The custom-lab container image uses jupyter/scipy-notebook as its base image and therefore has all its python packages (see https://github.com/jupyter/docker-stacks/blob/master/scipy-notebook/Dockerfile), but also installs the below additional python packages:
    - mysql-connector-python
    - matplotlib
    - tabulate
    - nbconvert
    - IPython
- NB: Packages are installed by using pip. This approach will require a user to install a package each time they log in. To have the packages installed by default, please see 7.4 Platform Maintenance Tutorial.
- The custom-lab container image also installs the jupyterlab git plugin upon startup. See https://github.com/jupyterlab/jupyterlab-git.

5.5 Database Implementation

Connection Information

The MySQL server runs in a container called 'db' as part of the platform Docker swarm. It is possible to connect directly to the database using any MySQL compatible client such as the mysql command line tool or MySQL workbench. MySQL software is free, open source and cross platform and can be downloaded from https://dev.mysql.com/downloads/

The connection runs on the (default) port 3306 and will be available at the IP address or Hostname of the computer running the Docker Platform

A DB admin user is created as part of the database setup. This user is defined in the platform file "mysql-db/sql-scripts/a_init_users.sql"
Default user and password are:

Username: 'dbmaintainer_user'

Password: 'dbmaintainer_password'

Application user with Write permissions:

Username: webapp_user

Password: webapp_password

JupyterHub application user with Read permissions:

Username: jupyter_user

Password: jupyter_password

5.5.1 Overview

During client discovery one of the key issues that was identified with the current audit process was the reliance on Excel spreadsheets for complex data analysis. These spreadsheets had grown extremely large in size making them slow and unstable to run, and complex to query. Storing data in spreadsheets carries risks and performance penalties such as: data errors due to a lack of integrity constraints, poor information security due to a lack of proper controls, difficulty with multiple user access over a network, inability to scale, and system Instability for large workbooks.

These risks were captured in user stories:

- 5 : Ensure data is secure at rest and in transit
- 6 : Have a stable system
- 7 : Extend upon the system

In addition, generating reports and extracting insights from this data required extensive use of complex and unintuitive macros and pivot tables to route and transform the data so that it could be analysed; as captured in user story 3 : "Design new approaches to analyse and audit data".

To address these user stories, we built a relational database using MySQL 8, modelling the “results”, and “results dose” tabs of the Level III audit workbook to bring the highly unnormalised data into a relational model of third normal form.

This enforces referential integrity, and eliminates redundancy, across the data set by providing constraints on data when it is entered, for example a specific Linac machine will be entered once in a look table and then referenced by a unique identifier in the results so that users can be confident when querying that they are referring to one specific machine and capturing all the relevant data points for that machine.

Additional metadata can also be captured about entities in the database such as the facility operators, their location, and the equipment, methods and algorithms used so that it becomes far easier to construct queries using SQL to see how certain providers, regions, or products compare in various audit metrics.

Having the data available in a relational database allows the system to be easily queried, analysed, visualised, and shared, in specialist tools such as the Jupyter Notebooks provided with this system while providing a separation of responsibilities between data storage, retrieval, and analysis.

While having the data normalised in this way provides advantages in terms of integrity and analysis it does make inserting data more onerous as order and integrity constraints matter when inserting data. To address this the system has a web application front end that allows for data to be inserted by either exporting it into CSV files and uploading through the web front end or programmatically through REST API calls. For querying the data three views audits_v, results_v, and facility_stated_v are provided to provide the data in a less normalised form where relevant tables are joined together. Additionally, user defined views can also be created as needed or queries saved in Jupyter notebooks for future / repeated use.

During client discovery one of the key issues that was identified with the current audit process was the reliance on Excel spreadsheets for complex data analysis. These spreadsheets had grown extremely large in size making them slow and unstable to run, and complex to query. Storing data in spreadsheets carries risks and performance penalties such as: data errors due to a lack of integrity constraints, poor information security due to a lack of proper controls, difficulty with multiple user access over a network, inability to scale, and system Instability for large workbooks.

These risks were captured in user stories:

- 5 : Ensure data is secure at rest and in transit
- 6 : Have a stable system
- 7 : Extend upon the system

In addition, generating reports and extracting insights from this data required extensive use of complex and unintuitive macros and pivot tables to route and transform the data so that it could be analysed; as captured in user story 3 : "Design new approaches to analyse and audit data".

To address these user stories, we built a relational database using MySQL 8, modelling the “results”, and “results dose” tabs of the Level III audit workbook to bring the highly unnormalised data into a relational model of third normal form.

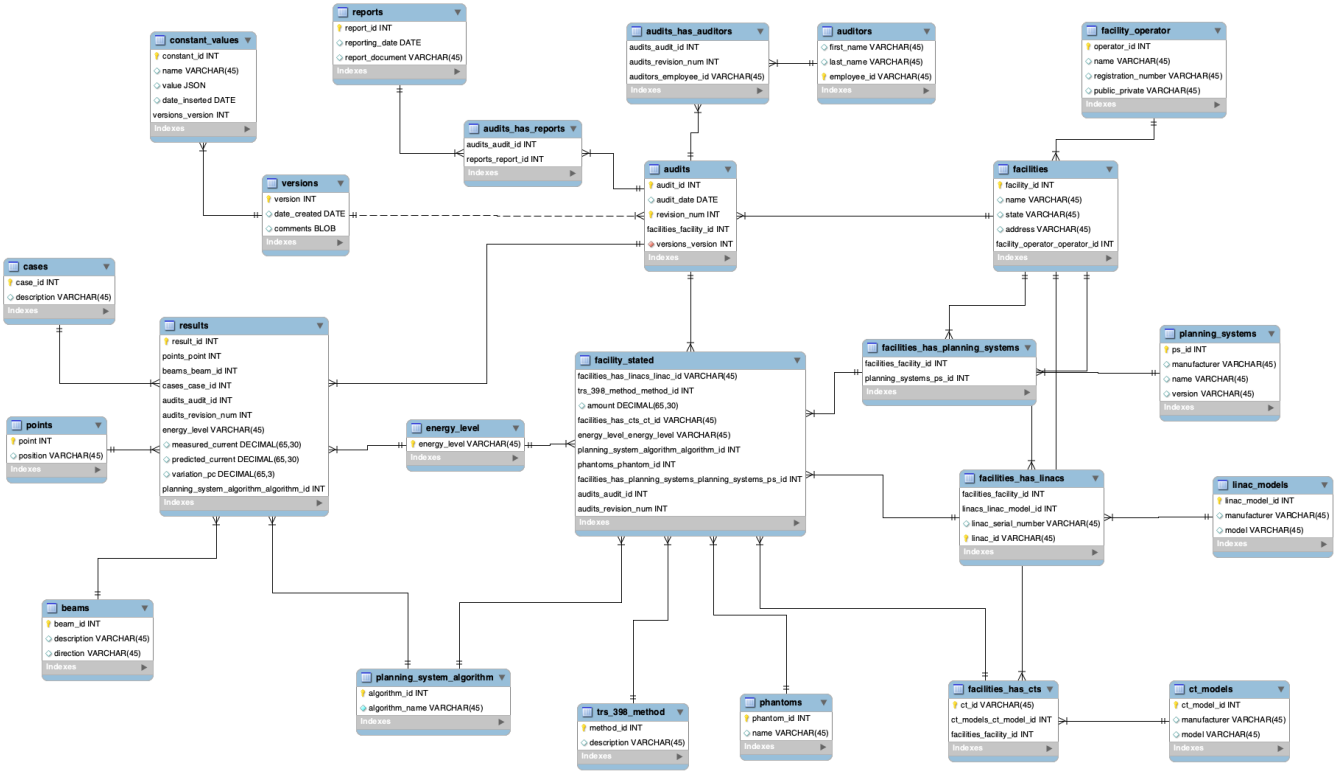
This enforces referential integrity, and eliminates redundancy, across the data set by providing constraints on data when it is entered, for example a specific Linac machine will be entered once in a look table and then referenced by a unique identifier in the results so that users can be confident when querying that they are referring to one specific machine and capturing all the relevant data points for that machine.

Additional metadata can also be captured about entities in the database such as the facility operators, their location, and the equipment, methods and algorithms used so that it becomes far easier to construct queries using SQL to see how certain providers, regions, or products compare in various audit metrics.

Having the data available in a relational database allows the system to be easily queried, analysed, visualised, and shared, in specialist tools such as the Jupyter Notebooks provided with this system while providing a separation of responsibilities between data storage, retrieval, and analysis.

While having the data normalised in this way provides advantages in terms of integrity and analysis it does make inserting data more onerous as order and integrity constraints matter when inserting data. To address this the system has a web application front end that allows for data to be inserted by either exporting it into CSV files and uploading through the web front end or programmatically through REST API calls. For querying the data three views audits_v, results_v, and facility_stated_v are provided to provide the data in a less normalised form where relevant tables are joined together. Additionally, user defined views can also be created as needed or queries saved in Jupyter notebooks for future / repeated use.

5.5.2 Entity Relationship Diagram



5.5.3 Data Dictionary

Data Description

Below is a data dictionary with a list item for each table and view, their fields, data mappings, meanings, and other important data modelling assumptions where not obvious.

Tables:

1. auditors

Field	Type	Null	Key
first_name	varchar(45)	YES	
last_name	varchar(45)	YES	
employee_id	varchar(45)	NO	PRI

Description: Lookup table for the ACDS employees undertaking the audit. Employee_id is a varchar to allow for potentially nonnumeric employee IDs to be used. Auditors must exist in this table to be added to audits.

1. **audits**

Field	Type	Null	Key
audit_id	int	NO	PRI
audit_date	date	YES	
revision_num	int	NO	PRI
facilities_facility_id	int	NO	PRI
versions_version	int	NO	MUL

Description: Contains details of each audit. The results and facility stated data for an Audit are uniquely identified by the combination of an Audit number and a revision_num.

Maps to columns A and B in the Results tab.

versions_version indicates which version of the constant values where used in the calculation of a particular audit.

2. **audits_has_auditors**

Field	Type	Null	Key
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI
auditors_employee_id	varchar(45)	NO	PRI

Description: Linking table to associate auditors by their employee_id with specific audits identified by both an audit number and revision number.

3. **audits_has_reports**

Field	Type	Null	Key

audits_audit_id	int	NO	PRI
reports_report_id	int	NO	PRI

Description: Linking table to associate reports to audits. Reports are only associated with an audit number and not a revision number as 1 audit will only produce 1 report even if the audit requires multiple revisions.

4. **beams**

Field	Type	Null	Key
beam_id	int	NO	PRI
description	varchar(45)	YES	
direction	varchar(45)	YES	

Description: A lookup table for the beam associated with a row in the results table. beam_id is a numerical identifier, description allows for text description for beams such as "IMRT" that are not numerical, direction is a text field that allows for human friendly descriptions and queries such as "Right Lateral".

5. **cases**

Field	Type	Null	Key
case_id	int	NO	PRI
description	varchar(45)	YES	

Description: A lookup table for the case associated with a row in the results table. Description provides an optional field for a human friendly description.

6. **constant_values**

Field	Type	Null	Key
constant_id	int	NO	PRI
name	varchar(45)	YES	
value	json	YES	
date_inserted	date	YES	
versions_version	int	NO	PRI

Description: A table to store versioned constant values that can be used in calculations when analysing data in Jupyter. These variables are defined by both a constant and a version number. The intention for this is if a scientific or regulatory body publish a value and then update it you can re-calculate the values in the audit, or use it to compare how results would have looked if different values had been used. the value field is a JSON so that any complex datatype, structure, or Python object can potentially be serialised and stored in this table.

7. **ct_models**

Field	Type	Null	Key
ct_model_id	int	NO	PRI
manufacturer	varchar(45)	YES	
model	varchar(45)	YES	

Description: A lookup table to store all potential CT machine makes and models that are used across facilities. This level of data normalisation makes it possible to analyze how a specific make or model compares across different audits.

8. **energy_level**

Field	Type	Null	Key
energy_level	varchar(45)	NO	PRI

Description: A lookup table to store all possible energy levels e.g. 6, 6FFF, 10, 10FFF that are used in results and facility stated prior data. Ensures referential integrity in that all energy level values must exist in this table and different spelling like 6fff do not exist in the data set.

9. **facilities**

Field	Type	Null	Key
facility_id	int	NO	PRI
name	varchar(45)	YES	
state	varchar(45)	YES	
address	varchar(45)	YES	
facility_operator_operator_id	int	NO	PRI

Description: A lookup table to store all the facilities that are being audited. The state field allows queries to be constructed by state, for the purposes of this data model New Zealand is treated as a state. Address fields could be added easily to allow more fine grained reporting if required such as health care planning regions or similar.

facility_operator_operator_id provides a link to operators of facilities such as state government health authorities or private providers.

10. **facilities_has_cts**

Field	Type	Null	Key
ct_id	varchar(45)	NO	PRI
ct_models_ct_model_id	int	NO	PRI
facilities_facility_id	int	NO	PRI

Description: Linking table to associate specific CT machines (individual instances of a particular make and model) to facilities.

11. **facilites_has_linacs**

Field	Type	Null	Key
facilities_facility_id	int	NO	PRI
linacs_linac_model_id	int	NO	PRI
linac_serial_number	varchar(45)	YES	
linac_id	varchar(45)	NO	PRI

Description: Linking table to associate specific Linac machines (individual instances of a particular make and model) to facilities.

12. **facilities_has_planning_systems**

Field	Type	Null	Key
facilities_facility_id	int	NO	PRI
planning_systems_ps_id	int	NO	PRI

Description: Linking table to associate specific planning systems (TPS) (individual instances of a particular make and model) to facilities.

13. **facility_operator**

Field	Type	Null	Key
operator_id	int	NO	PRI
name	varchar(45)	YES	
registration_number	varchar(45)	YES	UNI
public_private	varchar(45)	YES	

Description: A lookup table to store all the facilities operators that may run one or more facilities. registration_number is an optional field for the operators external, regulatory registration / license number. public_private provides an option to record whether that operator is in the public or private system so that public v private operators results can be compared. Other metadata about operators could be added here if required.

14. **facility_stated**

Field	Type	Null	Key
facilities_has_linacs_linac_id	varchar(45)	NO	PRI
trs_398_method_method_id	int	NO	PRI
amount	decimal(65,30)	YES	
facilities_has_cts_ct_id	varchar(45)	NO	PRI
energy_level_energy_level	varchar(45)	NO	PRI
planning_system_algorithm_algorithm_id	int	NO	PRI
phantoms_phantom_id	int	NO	PRI
facilities_has_planning_systems_planning_systems_ps_id	int	NO	PRI
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI

Description: Facility stated is one of the main tables along with results to store audit data. All of the prior data provided by the facility on the day of the audit corresponding to rows AH to AV of the results tab of the spreadsheet is stored in this table. This table also records which specific Linac, CT Machine, TPS Planning System, Planning system algorithm, and Phantom was used in a given audit / revision combination. The trs_398 method describes whether the data point relates to a Daily Output, TPR 10,20, or Kq value, at a given energy. Amount is the quantitate value assigned to that entry.

15. **linac_models**

Field	Type	Null	Key
linac_model_id	int	NO	PRI
manufacturer	varchar(45)	YES	
model	varchar(45)	YES	

Description: A lookup table to store all potential Linac machine makes and models that are used across facilities. This level of data normalisation makes it possible to analyze how a specific make or model compares across different audits.

16. **phantoms**

Field	Type	Null	Key
phantom_id	int	NO	PRI
name	varchar(45)	YES	

Description: A lookup table to store all phantoms that are used across audits.

17. **planning_system_algorithm**

Field	Type	Null	Key
algorithm_id	int	NO	PRI
algorithm_name	varchar(45)	NO	UNI

Description: A lookup table to store all possible planning system algorithms. Corresponds to column X on the Results tab of the workbook.

18. **planning_systems**

Field	Type	Null	Key
ps_id	int	NO	PRI

manufacturer	varchar(45)	YES	
name	varchar(45)	YES	
version	varchar(45)	YES	

Description: A lookup table to store all possible planning systems / software used at various sites and their manufacturer, name and software version.

Instances of which planning systems are used at which sites is recorded in facilities_has_planning_systems.

19. **points**

Field	Type	Null	Key
point	int	NO	PRI
position	varchar(45)	YES	

Description: A lookup table to store all possible points on a phantom where case data could be measured from. The table ensures referential integrity across data points. Position is an optional text field to store a human friendly description of the point's anatomical position to assist in querying and analysing data.

20. **reports**

Field	Type	Null	Key
report_id	int	NO	PRI
reporting_date	date	YES	
report_document	varchar(45)	YES	

Description: A lookup table to record the report document(s), these are linked to a specific audit number through the linking table audits_has_reports. report_document text filed can store a document name or link to a document on a network share.

21. **results**

Field	Type	Null	Key
result_id	int	NO	PRI
points_point	int	NO	PRI
beams_beam_id	int	NO	PRI
cases_case_id	int	NO	PRI
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI
energy_level	varchar(45)	NO	PRI
measured_current	decimal(65,30)	YES	
predicted_current	decimal(65,30)	YES	
variation_pc	decimal(65,3)	YES	
planning_system_algorithm_algorithm_id	int	NO	PRI

Description: The results table is one of the main tables along with facility stated for storing audit data. Results stores the case data / measurements taken from the phantom. Each row should be a unique combination of a point, beam, case and energy level taken on a given audit / revision. These fields along with the planning system algorithm used are all foreign keys to enforce referential integrity in the data. Each row has the option to store a variation percentage (variation_pc) , predicted current, and measured current. This way the table captures the data (optionally) of both results and results dose

Result_id is a numeric identifier given to each result entry and is not dependant on any other tables.

The view results_v is provided for querying results data more efficiently and provides the data in a less normalised form where the foreign keys and lookup values are all joined to the results data.

22. **versions**

Field	Type	Null	Key
version	int	NO	PRI
date_created	date	YES	
comments	blob	YES	

Description: Table to describe which version of the constant values has been used in the calculations of a given audit. Each audit contains a version number, and each constant value has a corresponding version number so that by looking at an audit you can determine what version of the constants where used.

Views:

Three views are provided that link results to the associated metadata and look up tables to make querying easier.

1. **audits_v**

Field	Type	Null
audit_id	int	NO
revision_num	int	NO
versions_version	int	NO
audit_date	date	YES
facilities_facility_id	int	NO
facility_name	varchar(45)	YES
facility_operator_name	varchar(45)	YES
operator_id	int	NO
report_id	int	NO
reporting_date	date	YES

Description: A view to make querying the audits table easier. This view performs inner joins on audit, version, facility, facility operator, and report tables.

Sample Data:

audit_id	revision_num	versions_version	audit_date	facilities_facility...	facility_name	facility_operator_name	operator_id	report_id	reporting_date
3081	1	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	1	2017-01-01
3081	2	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	1	2017-01-01
3110	1	1	2019-07-01	48	Akita	Operator A	1	1	2017-01-01
3150	1	1	2018-09-09	41	Badger	Operator C	3	1	2017-01-01
3151	1	1	2019-02-02	13	American Water Spaniel	Operator D	4	1	2017-01-01
3151	2	1	2019-03-01	13	American Water Spaniel	Operator D	4	1	2017-01-01
3173	1	1	2019-08-02	90	Boykin Spaniel	Operator D	4	1	2017-01-01
3200	1	1	2017-07-05	5	Ainu Dog	Operator A	1	1	2017-01-01
3204	1	1	2020-03-13	13	American Water Spaniel	Operator D	4	1	2017-01-01
3207	1	1	2018-06-11	24	Barracuda	Operator A	1	1	2017-01-01
3654	1	1	2018-01-01	41	Badger	Operator C	3	1	2017-01-01
3081	1	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	2	2017-01-01
3081	2	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	2	2017-01-01

2. **facility_stated_v**

Field	Type	Null
audit_id	int	NO
revision_num	int	NO
audit_date	date	YES

facility_id	int	NO
facility_name	varchar(45)	YES
linac_id	varchar(45)	NO
linac_serial_number	varchar(45)	YES
linac_model	varchar(45)	YES
linac_manufacturer	varchar(45)	YES
trs_398_id	int	NO
trs_398_description	varchar(45)	YES
planning_system_id	int	NO
planning_system_name	varchar(45)	YES
planning_system_version	varchar(45)	YES
planning_system_manufacturer	varchar(45)	YES
ps_algorithm_id	int	NO
ps_algorithm_name	varchar(45)	NO
ct_id	varchar(45)	NO
ct_model	varchar(45)	YES
ct_manufacturer	varchar(45)	YES
phantom_id	int	NO
phantom_name	varchar(45)	YES
energy_level_energy_level	varchar(45)	NO
amount	decimal(65,30)	YES

Description: consolidates the data by joining tables linked to the facility stated table. Joined tables are:
audits, facilities, facilities_has_linacs, linac_models, trs_398_method, planning_systems, planning_system_algorithm, ct_models, phantoms.

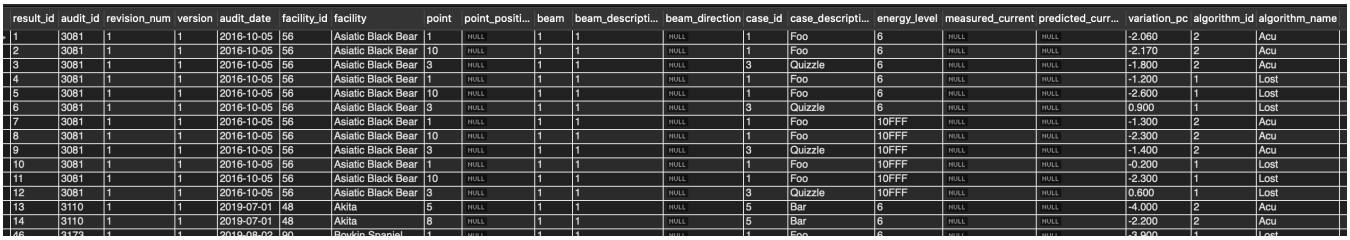
audit_id	revision_num	audit_date	facility_id	facility_name	linac_id	linac_serial_number	linac_model	linac_manufactur...	trs_398_id	trs_398_description	planning_system_id	planning_system_name	planning_system_version	planning_system_manufactu...	ps_algorithm_id	ps_algorithm_name	ct_id	ct_model	ct_manufactur...	phantom_id	phantom_name	energy_level_energy_level	amount
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	1.0200000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	1.0030000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	1.0020000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	1.0020000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	0.7070000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	0.7070000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	0.6670000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	0.6670000000000000000000000000
3151	1	2019-02-02	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	1	Daily Output (cGy/MU)	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	1.8900000000000000000000000000
3151	1	2019-02-02	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	2	TPR20,10	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	1.9950000000000000000000000000
3151	2	2019-03-01	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	1	Daily Output (cGy/MU)	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	2.0000000000000000000000000000
3151	2	2019-03-01	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	2	TPR20,10	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	2.1000000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	1.0035867500000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	1.0870000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	1.8890000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	1.9260000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	0.5700000000000000000000000000

3. **results_v**

Field	Type	Null
result_id	int	NO
audit_id	int	NO
revision_num	int	NO
version	int	NO
audit_date	date	YES
facility_id	int	NO
facility	varchar(45)	YES
point	int	NO
point_position	varchar(45)	YES
beam	int	NO

beam_description	varchar(45)	YES
beam_direction	varchar(45)	YES
case_id	int	NO
case_description	varchar(45)	YES
energy_level	varchar(45)	NO
measured_current	decimal(65,30)	YES
predicted_current	decimal(65,30)	YES
variation_pc	decimal(65,3)	YES
algorithm_id	int	NO

Description: consolidates the data by joining tables linked to the results table. Joined tables are: points, beams, cases, audits, facilities, and planning_system_algorithm



result_id	audit_id	revision_num	version	audit_date	facility_id	facility	point	point_positi...	beam	beam_descripti...	beam_direction	case_id	case_descripti...	energy_level	measured_current	predicted_curr...	variation_pc	algorithm_id	algorithm_name
1	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.060	2	Acu
2	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.170	2	Acu
3	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	6	NULL	NULL	-1.800	2	Acu
4	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-1.200	1	Lost
5	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.600	1	Lost
6	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	6	NULL	NULL	0.900	1	Lost
7	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-1.300	2	Acu
8	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-2.300	2	Acu
9	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	10FFF	NULL	NULL	-1.400	2	Acu
10	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-0.200	1	Lost
11	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-2.300	1	Lost
12	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	10FFF	NULL	NULL	0.600	1	Lost
13	3110	1	1	2019-07-01	48	Akita	5	NULL	1	1	NULL	5	Bar	6	NULL	NULL	-4.000	2	Acu
14	3110	1	1	2019-07-01	48	Akita	8	NULL	1	1	NULL	5	Bar	6	NULL	NULL	-2.200	2	Acu
46	3173	1	1	2019-08-02	90	Boykin Spaniel	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-3.900	1	Lost

Data Description

Below is a data dictionary with a list item for each table and view, their fields, data mappings, meanings, and other important data modelling assumptions where not obvious.

Tables:

1. auditors

Field	Type	Null	Key
first_name	varchar(45)	YES	
last_name	varchar(45)	YES	
employee_id	varchar(45)	NO	PRI

Description: Lookup table for the ACDS employees undertaking the audit. Employee_id is a varchar to allow for potentially nonnumeric employee IDs to be used. Auditors must exist in this table to be added to audits.

1. **audits**

Field	Type	Null	Key
audit_id	int	NO	PRI
audit_date	date	YES	
revision_num	int	NO	PRI
facilities_facility_id	int	NO	PRI
versions_version	int	NO	MUL

Description: Contains details of each audit. The results and facility stated data for an Audit are uniquely identified by the combination of an Audit number and a revision_num.

Maps to columns A and B in the Results tab.

versions_version indicates which version of the constant values where used in the calculation of a particular audit.

2. **audits_has_auditors**

Field	Type	Null	Key
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI
auditors_employee_id	varchar(45)	NO	PRI

Description: Linking table to associate auditors by their employee_id with specific audits identified by both an audit number and revision number.

3. **audits_has_reports**

Field	Type	Null	Key
audits_audit_id	int	NO	PRI
reports_report_id	int	NO	PRI

Description: Linking table to associate reports to audits. Reports are only associated with an audit number and not a revision number as 1 audit will only produce 1 report even if the audit requires multiple revisions.

4. **beams**

Field	Type	Null	Key
beam_id	int	NO	PRI
description	varchar(45)	YES	
direction	varchar(45)	YES	

Description: A lookup table for the beam associated with a row in the results table. beam_id is a numerical identifier, description allows for text description for beams such as "IMRT" that are not numerical, direction is a text field that allows for human friendly descriptions and queries such as "Right Lateral".

5. **cases**

Field	Type	Null	Key
case_id	int	NO	PRI
description	varchar(45)	YES	

Description: A lookup table for the case associated with a row in the results table. Description provides an optional field for a human friendly description.

6. **constant_values**

Field	Type	Null	Key
constant_id	int	NO	PRI
name	varchar(45)	YES	
value	json	YES	
date_inserted	date	YES	

versions_version	int	NO	PRI

Description: A table to store versioned constant values that can be used in calculations when analysing data in Jupyter. These variables are defined by both a constant and a version number. The intention for this is if a scientific or regulatory body publish a value and then update it you can re-calculate the values in the audit, or use it to compare how results would have looked if different values had been used. the value field is a JSON so that any complex datatype, structure, or Python object can potentially be serialised and stored in this table.

7. **ct_models**

Field	Type	Null	Key
ct_model_id	int	NO	PRI
manufacturer	varchar(45)	YES	
model	varchar(45)	YES	

Description: A lookup table to store all potential CT machine makes and models that are used across facilities. This level of data normalisation makes it possible to analyze how a specific make or model compares across different audits.

8. **energy_level**

Field	Type	Null	Key
energy_level	varchar(45)	NO	PRI

Description: A lookup table to store all possible energy levels e.g. 6, 6FFF, 10, 10FFF that are used in results and facility stated prior data. Ensures referential integrity in that all energy level values must exist in this table and different spelling like 6fff do not exist in the data set.

9. **facilities**

Field	Type	Null	Key
facility_id	int	NO	PRI
name	varchar(45)	YES	
state	varchar(45)	YES	
address	varchar(45)	YES	
facility_operator_operator_id	int	NO	PRI

Description: A lookup table to store all the facilities that are being audited. The state field allows queries to be constructed by state, for the purposes of this data model New Zealand is treated as a state. Address fields could be added easily to allow more fine grained reporting if required such as health care planning regions or similar.

facility_operator_operator_id provides a link to operators of facilities such as state government health authorities or private providers.

10. **facilities_has_cts**

Field	Type	Null	Key
ct_id	varchar(45)	NO	PRI
ct_models_ct_model_id	int	NO	PRI
facilities_facility_id	int	NO	PRI

Description: Linking table to associate specific CT machines (individual instances of a particular make and model) to facilities.

11. **facilites_has_linacs**

Field	Type	Null	Key
facilities_facility_id	int	NO	PRI
linacs_linac_model_id	int	NO	PRI

linac_serial_number	varchar(45)	YES	
linac_id	varchar(45)	NO	PRI

Description: Linking table to associate specific Linac machines (individual instances of a particular make and model) to facilities.

12. **facilities_has_planning_systems**

Field	Type	Null	Key
facilities_facility_id	int	NO	PRI
planning_systems_ps_id	int	NO	PRI

Description: Linking table to associate specific planning systems (TPS) (individual instances of a particular make and model) to facilities.

13. **facility_operator**

Field	Type	Null	Key
operator_id	int	NO	PRI
name	varchar(45)	YES	
registration_number	varchar(45)	YES	UNI
public_private	varchar(45)	YES	

Description: A lookup table to store all the facilities operators that may run one or more facilities. registration_number is an optional field for the operators external, regulatory registration / license number. public_private provides an option to record whether that operator is in the public or private system so that public v private operators results can be compared. Other metadata about operators could be added here if required.

14. **facility_stated**

Field	Type	Null	Key
facilities_has_linacs_linac_id	varchar(45)	NO	PRI
trs_398_method_method_id	int	NO	PRI
amount	decimal(65,30)	YES	
facilities_has_cts_ct_id	varchar(45)	NO	PRI
energy_level_energy_level	varchar(45)	NO	PRI
planning_system_algorithm_algorithm_id	int	NO	PRI
phantoms_phantom_id	int	NO	PRI
facilities_has_planning_systems_planning_systems_ps_id	int	NO	PRI
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI

Description: Facility stated is one of the main tables along with results to store audit data. All of the prior data provided by the facility on the day of the audit corresponding to rows AH to AV of the results tab of the spreadsheet is stored in this table. This table also records which specific Linac, CT Machine, TPS Planning System, Planning system algorithm, and Phantom was used in a given audit / revision combination. The trs_398 method describes whether the data point relates to a Daily Output, TPR 10,20, or Kq value, at a given energy. Amount is the quantitate value assigned to that entry.

15. **linac_models**

Field	Type	Null	Key
linac_model_id	int	NO	PRI
manufacturer	varchar(45)	YES	
model	varchar(45)	YES	

Description: A lookup table to store all potential Linac machine makes and models that are used across facilities. This level of data normalisation makes it possible to analyze how a specific make or model compares across different audits.

16. **phantoms**

Field	Type	Null	Key
phantom_id	int	NO	PRI
name	varchar(45)	YES	

Description: A lookup table to store all phantoms that are used across audits.

17. **planning_system_algorithm**

Field	Type	Null	Key
algorithm_id	int	NO	PRI
algorithm_name	varchar(45)	NO	UNI

Description: A lookup table to store all possible planning system algorithms. Corresponds to column X on the Results tab of the workbook.

18. **planning_systems**

Field	Type	Null	Key
ps_id	int	NO	PRI
manufacturer	varchar(45)	YES	
name	varchar(45)	YES	
version	varchar(45)	YES	

Description: A lookup table to store all possible planning systems / software used at various sites and their manufacturer, name and software version.

Instances of which planning systems are used at which sites is recorded in facilities_has_planning_systems.

19. **points**

Field	Type	Null	Key
point	int	NO	PRI
position	varchar(45)	YES	

Description: A lookup table to store all possible points on a phantom where case data could be measured from. The table ensures referential integrity across data points. Position is an optional text field to store a human friendly description of the point's anatomical position to assist in querying and analysing data.

20. **reports**

Field	Type	Null	Key
report_id	int	NO	PRI
reporting_date	date	YES	
report_document	varchar(45)	YES	

Description: A lookup table to record the report document(s), these are linked to a specific audit number through the linking table audits_has_reports. report_document text filed can store a document name or link to a document on a network share.

21. **results**

Field	Type	Null	Key
result_id	int	NO	PRI
points_point	int	NO	PRI

beams_beam_id	int	NO	PRI
cases_case_id	int	NO	PRI
audits_audit_id	int	NO	PRI
audits_revision_num	int	NO	PRI
energy_level	varchar(45)	NO	PRI
measured_current	decimal(65,30)	YES	
predicted_current	decimal(65,30)	YES	
variation_pc	decimal(65,3)	YES	
planning_system_algorithm_algorithm_id	int	NO	PRI

Description: The results table is one of the main tables along with facility stated for storing audit data. Results stores the case data / measurements taken from the phantom. Each row should be a unique combination of a point, beam, case and energy level taken on a given audit / revision. These fields along with the planning system algorithm used are all foreign keys to enforce referential integrity in the data. Each row has the option to store a variation percentage (variation_pc) , predicted current, and measured current. This way the table captures the data (optionally) of both results and results dose

Result_id is a numeric identifier given to each result entry and is not dependant on any other tables.

The view results_v is provided for querying results data more efficiently and provides the data in a less normalised form where the foreign keys and lookup values are all joined to the results data.

22. **versions**

Field	Type	Null	Key
version	int	NO	PRI
date_created	date	YES	
comments	blob	YES	

Description: Table to describe which version of the constant values has been used in the calculations of a given audit. Each audit contains a version number, and each constant value has a corresponding version number so that by looking at an audit you can determine what version of the constants where used.

Views:

Three views are provided that link results to the associated metadata and look up tables to make querying easier.

1. **audits_v**

Field	Type	Null
audit_id	int	NO
revision_num	int	NO
versions_version	int	NO
audit_date	date	YES
facilities_facility_id	int	NO
facility_name	varchar(45)	YES
facility_operator_name	varchar(45)	YES
operator_id	int	NO
report_id	int	NO
reporting_date	date	YES

Description: A view to make querying the audits table easier. This view performs inner joins on audit, version, facility, facility operator, and report tables.

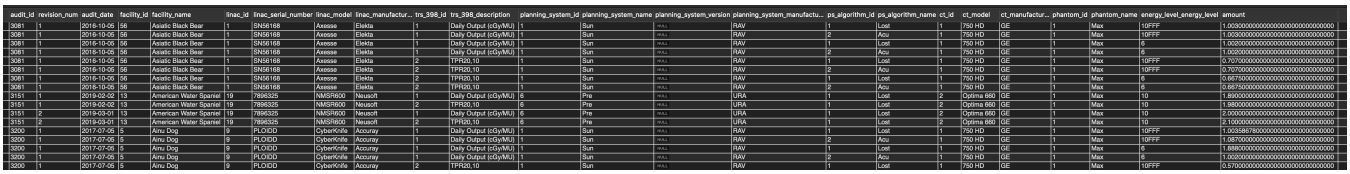
Sample Data:

audit_id	revision_num	versions_version	audit_date	facilities_facility...	facility_name	facility_operator_name	operator_id	report_id	reporting_date
3081	1	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	1	2017-01-01
3081	2	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	1	2017-01-01
3110	1	1	2019-07-01	48	Akita	Operator A	1	1	2017-01-01
3150	1	1	2018-09-09	41	Badger	Operator C	3	1	2017-01-01
3151	1	1	2019-02-02	13	American Water Spaniel	Operator D	4	1	2017-01-01
3151	2	1	2019-03-01	13	American Water Spaniel	Operator D	4	1	2017-01-01
3173	1	1	2019-08-02	90	Boykin Spaniel	Operator D	4	1	2017-01-01
3200	1	1	2017-07-05	5	Ainu Dog	Operator A	1	1	2017-01-01
3204	1	1	2020-03-13	13	American Water Spaniel	Operator D	4	1	2017-01-01
3207	1	1	2018-06-11	24	Barracuda	Operator A	1	1	2017-01-01
3654	1	1	2018-01-01	41	Badger	Operator C	3	1	2017-01-01
3081	1	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	2	2017-01-01
3081	2	1	2016-10-05	56	Asiatic Black Bear	Operator A	1	2	2017-01-01

2. **facility_stated_v**

Field	Type	Null
audit_id	int	NO
revision_num	int	NO
audit_date	date	YES
facility_id	int	NO
facility_name	varchar(45)	YES
linac_id	varchar(45)	NO
linac_serial_number	varchar(45)	YES
linac_model	varchar(45)	YES
linac_manufacturer	varchar(45)	YES
trs_398_id	int	NO
trs_398_description	varchar(45)	YES
planning_system_id	int	NO
planning_system_name	varchar(45)	YES
planning_system_version	varchar(45)	YES
planning_system_manufacturer	varchar(45)	YES
ps_algorithm_id	int	NO
ps_algorithm_name	varchar(45)	NO
ct_id	varchar(45)	NO
ct_model	varchar(45)	YES
ct_manufacturer	varchar(45)	YES
phantom_id	int	NO
phantom_name	varchar(45)	YES
energy_level_energy_level	varchar(45)	NO
amount	decimal(65,30)	YES

Description: consolidates the data by joining tables linked to the facility stated table. Joined tables are:
audits, facilities, facilities_has_linacs, linac_models, trs_398_method, planning_systems, planning_system_algorithm, ct_models, phantoms.



audit_id	revision_num	audit_date	facility_id	facility_name	linac_id	linac_serial_number	linac_model	linac_manufactur...	trs_398_id	trs_398_description	planning_system_id	planning_system_name	planning_system_version	planning_system_manufactu...	ps_algorithm_id	ps_algorithm_name	ct_id	ct_model	ct_manufactur...	phantom_id	phantom_name	energy_level_energy_level	amount
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	1,0030000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	1,0030000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	1,0020000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	1,0020000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	0,7070000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	0,7070000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	0,6675000000000000000000000000
3081	1	2016-10-05	56	Asiatic Black Bear	1	SN56168	Axesse	Elekta	2	TPR20,10	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	0,6675000000000000000000000000
3151	1	2019-02-02	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	1	Daily Output (cGy/MU)	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	1,8900000000000000000000000000
3151	1	2019-02-02	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	2	TPR20,10	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	1,9800000000000000000000000000
3151	2	2019-03-01	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	1	Daily Output (cGy/MU)	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	2,0000000000000000000000000000
3151	2	2019-03-01	13	American Water Spaniel	19	7896325	NMSR600	Neusoft	2	TPR20,10	6	Pre	NULL	URA	1	Lost	2	Optima 660	GE	1	Max	10	2,1000000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	1,0035867800000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	10FFF	1,0870000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	6	1,8880000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	1	Daily Output (cGy/MU)	1	Sun	NULL	RAV	2	Acu	1	750 HD	GE	1	Max	6	1,0020000000000000000000000000
3200	1	2017-07-05	5	Ainu Dog	9	PLOIDD	CyberKnife	Accuray	2	TPR20,10	1	Sun	NULL	RAV	1	Lost	1	750 HD	GE	1	Max	10FFF	0,5700000000000000000000000000

3. **results_v**

Field	Type	Null
result_id	int	NO
audit_id	int	NO
revision_num	int	NO
version	int	NO
audit_date	date	YES
facility_id	int	NO
facility	varchar(45)	YES
point	int	NO
point_position	varchar(45)	YES
beam	int	NO
beam_description	varchar(45)	YES
beam_direction	varchar(45)	YES
case_id	int	NO
case_description	varchar(45)	YES
energy_level	varchar(45)	NO
measured_current	decimal(65,30)	YES
predicted_current	decimal(65,30)	YES
variation_pc	decimal(65,3)	YES
algorithm_id	int	NO

Description: consolidates the data by joining tables linked to the results table. Joined tables are: points, beams, cases, audits, facilities, and planning_system_algorithm

result_id	audit_id	revision_num	version	audit_date	facility_id	facility	point	point_positi...	beam	beam_descripti...	beam_direction	case_id	case_descripti...	energy_level	measured_current	predicted_curr...	variation_pc	algorithm_id	algorithm_name
1	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.060	2	Acu
2	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.170	2	Acu
3	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	6	NULL	NULL	-1.800	2	Acu
4	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-1.200	1	Lost
5	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-2.600	1	Lost
6	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	6	NULL	NULL	0.900	1	Lost
7	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-1.300	2	Acu
8	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-2.300	2	Acu
9	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	10FFF	NULL	NULL	-1.400	2	Acu
10	3081	1	1	2016-10-05	56	Asiatic Black Bear	1	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-0.200	1	Lost
11	3081	1	1	2016-10-05	56	Asiatic Black Bear	10	NULL	1	1	NULL	1	Foo	10FFF	NULL	NULL	-2.300	1	Lost
12	3081	1	1	2016-10-05	56	Asiatic Black Bear	3	NULL	1	1	NULL	3	Quizzle	10FFF	NULL	NULL	0.600	1	Lost
13	3110	1	1	2019-07-01	48	Akita	5	NULL	1	1	NULL	5	Bar	6	NULL	NULL	-4.000	2	Acu
14	3110	1	1	2019-07-01	48	Akita	8	NULL	1	1	NULL	5	Bar	6	NULL	NULL	-2.200	2	Acu
146	3173	1	1	2019-08-02	90	Boykin Spaniel	1	NULL	1	1	NULL	1	Foo	6	NULL	NULL	-3.900	1	Lost

6 Tutorials

This section contains tutorials detailing how to use every aspect of our system. It includes instructions on setting up the system on a machine and the process of inserting or retrieving data from the database.

Contents

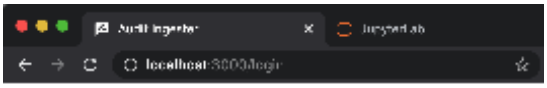
- 6.1 Data Ingestion Tutorial
- 6.2 Data Preparation Tutorial
- 6.3 Jupyter Tutorial
- 6.4 Software (Platform) Deployment & Maintenance Tutorial

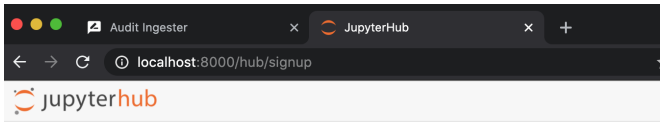
6.1 Data Ingestion Tutorial

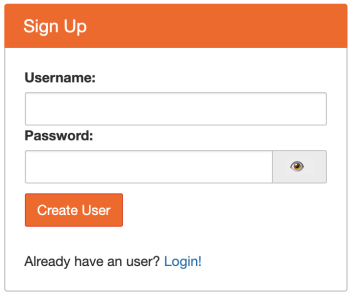
The system requires 3 files be provided for the audit: meta.json, facilty_stated.csv, and results.csv. Please see the guide in section 6.2 on how to prepare this data, or use the examples provided before beginning this guide.

Tutorial:

1. Navigate to the "Audit Ingester" web front end. This will be available on port 3000 of the server that is running the Platform Docker swarm such as "http://<server_address>:3000" . since I am running this on my local computer the address is http://localhost:3000. You will be asked to log in using the username and password that was created in Jupyter Labs, if you do not already have a username and password then please create one on the Jupyter hub login page which is available on the same server at port 8000 e.g. "http://<server_address>:8000".



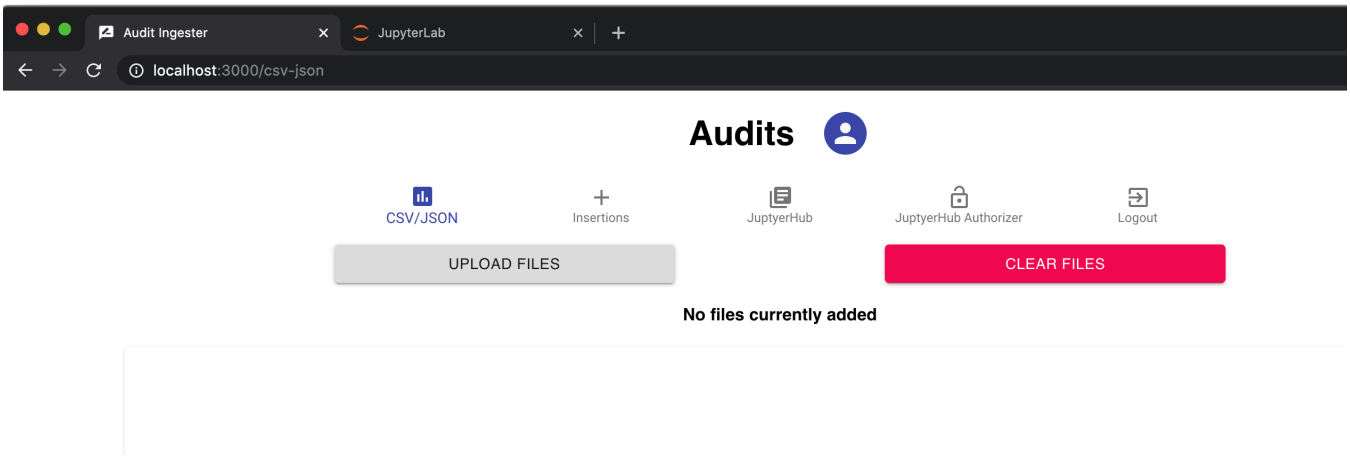




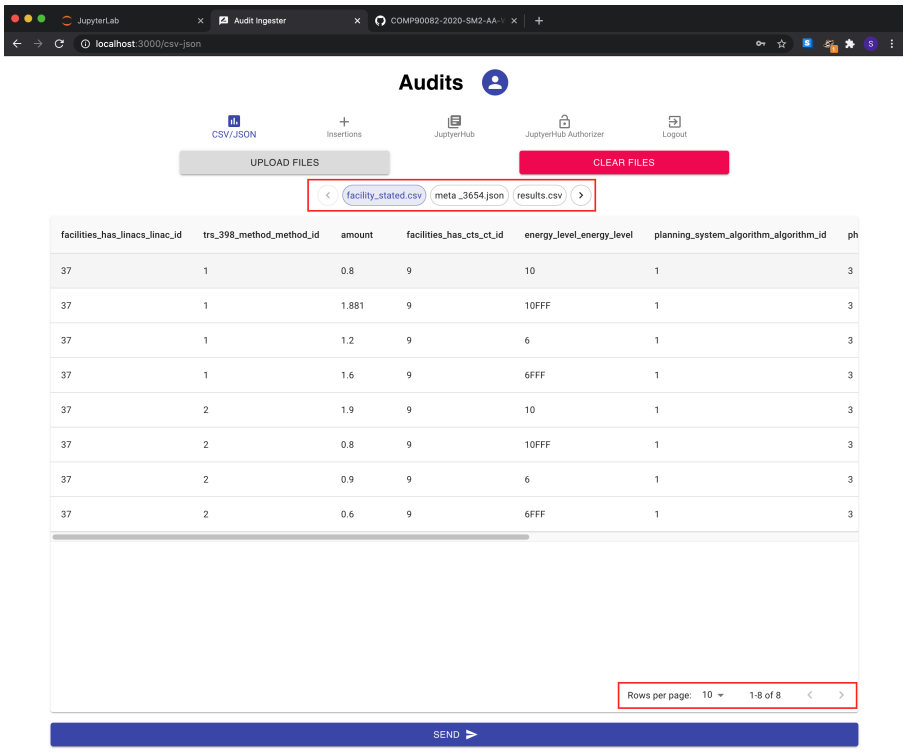
2. After logging in you should be redirected to the main page, ensuring that you have the CSV/JSON button selected click on "upload files". This button allows you to upload the files created in section 7.4. from your local computer.

You can upload one copy each of the files to be inserted into the database. As long as the foreign key constraints are satisfied, it is not necessary to upload all of the file at once. For example you could create the records for a specific audit by uploading the meta.json on one day, then providing the facility_stated.csv, and results.csv the next day after completing the audit.

For this tutorial our file contains data for one audit, but there is no limit to the number of audits that can be included within those three files, provided database key constraints are satisfied when inserting data.



3. For this example we will load will three files that describe the audit "3654" revision 1.

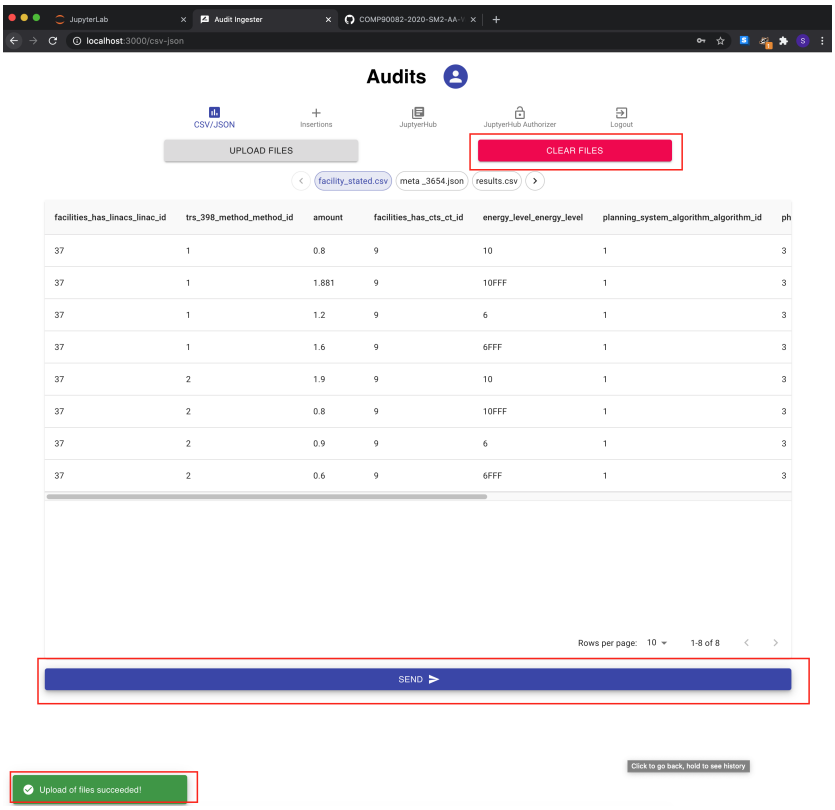


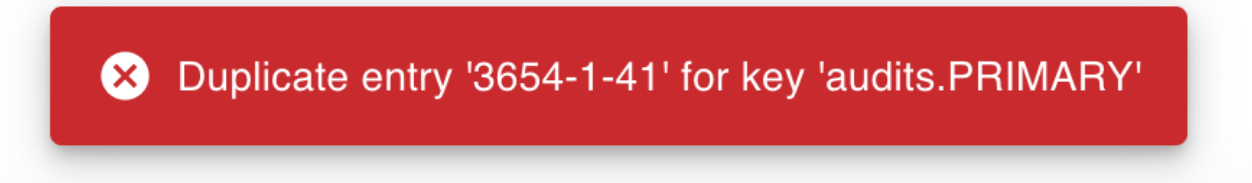
After uploading the files the system will read them and display the contents that it will insert into the database so that it can be verified before writing into the database.

The bar at the top (highlighted by the red box) shows the files that have been uploaded, you can scroll between each of the files to see a preview of the contents of each file. In the bottom right corner (also highlighted in red) are the pagination controls, for longer files it maybe necessary to page through them or scroll to see the entire contents.

4. If you are happy with the contents of the file then press the blue "send" bar at the bottom of the screen (highlighted in red) to write the data to the database. Upon writing to the database notification (highlighted in red) will slide in on the bottom left of the screen to let you know if the write was successful. If there are any errors then the notification will be red and contain the error message. The errors are SQL errors, passed up from the database and will most commonly be for key constraints, duplicates or incompatible values. If you receive an error you will need to correct it by clearing the files, correcting any errors in the file and re-uploading.

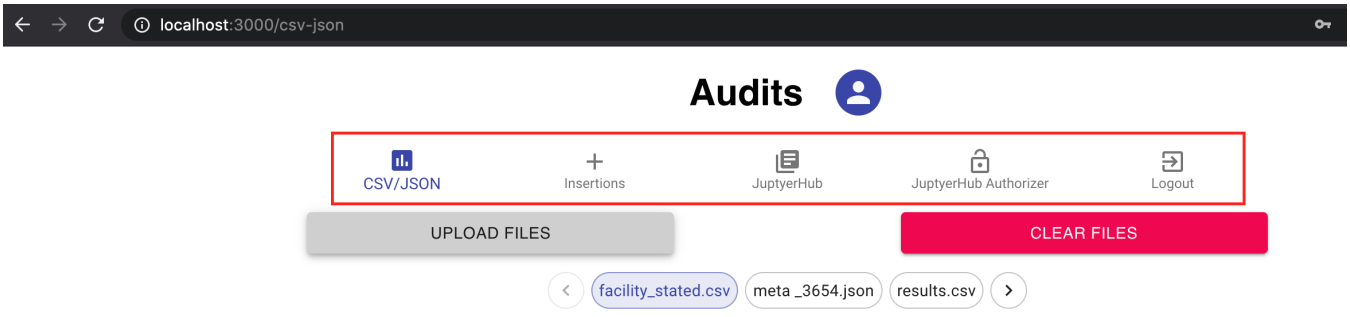
If you are not happy with the uploaded files and would like to cancel without writing into the database then you can do this by clicking the red "Clear Files" button.



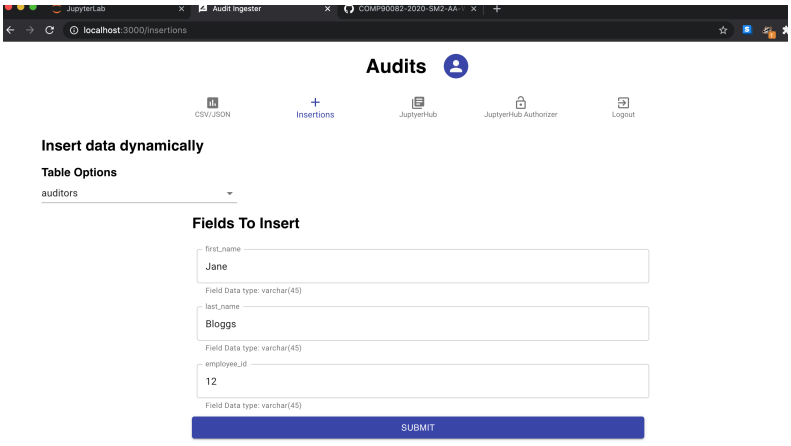


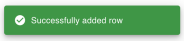
5. After writing the to the database and receiving your status information you have four options: Either clear the files and repeat the process to upload more data, use the inserts option to manually write more data to the database, click the JupyterHub button to analyse the data, or logout to end your session.

More information on performing analysis and generating reports with Jupyter can be found in Section 7.5 Jupyter Hub Tutorial.



6. For this tutorial we will use the insertions button (with the + sign) to add a new employee as an auditor and manually assign them to this audit, in addition to the 2 auditors that were specified in the meta.json file.





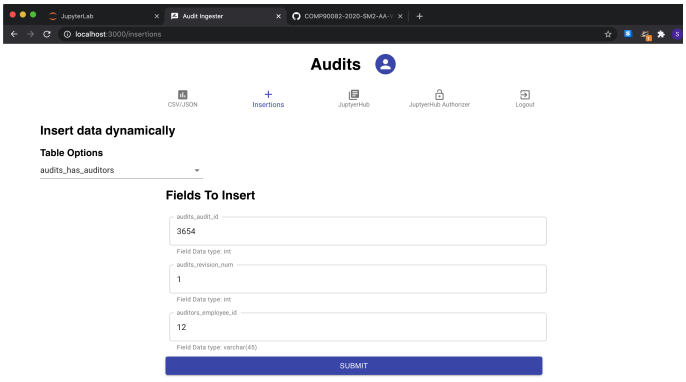
Information about ACDS employees who are performing audits is stored in the 'auditors' table. We can select the auditors table from the list of all available tables in the database from the drop down menu under "Insert data dynamically" - table options.

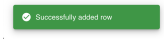
After selecting the table the page will update with all the fields (columns) in this table. This section, along with the table list are reflected dynamically from the database so should reflect any changes made to the underlying database automatically.

After entering the new employees information we can press the send button to write to the database and receive a notification to let us know whether or not the data was successfully inserted into the database.

7. Since many auditors will be working on many audits we will need to associate this auditor with the employee ID "12" to our audit ID "3654" with revision number "1".

This linking is done in the in the table"audits_has_auditors" which we can select in the same way form the "table options" drop down.





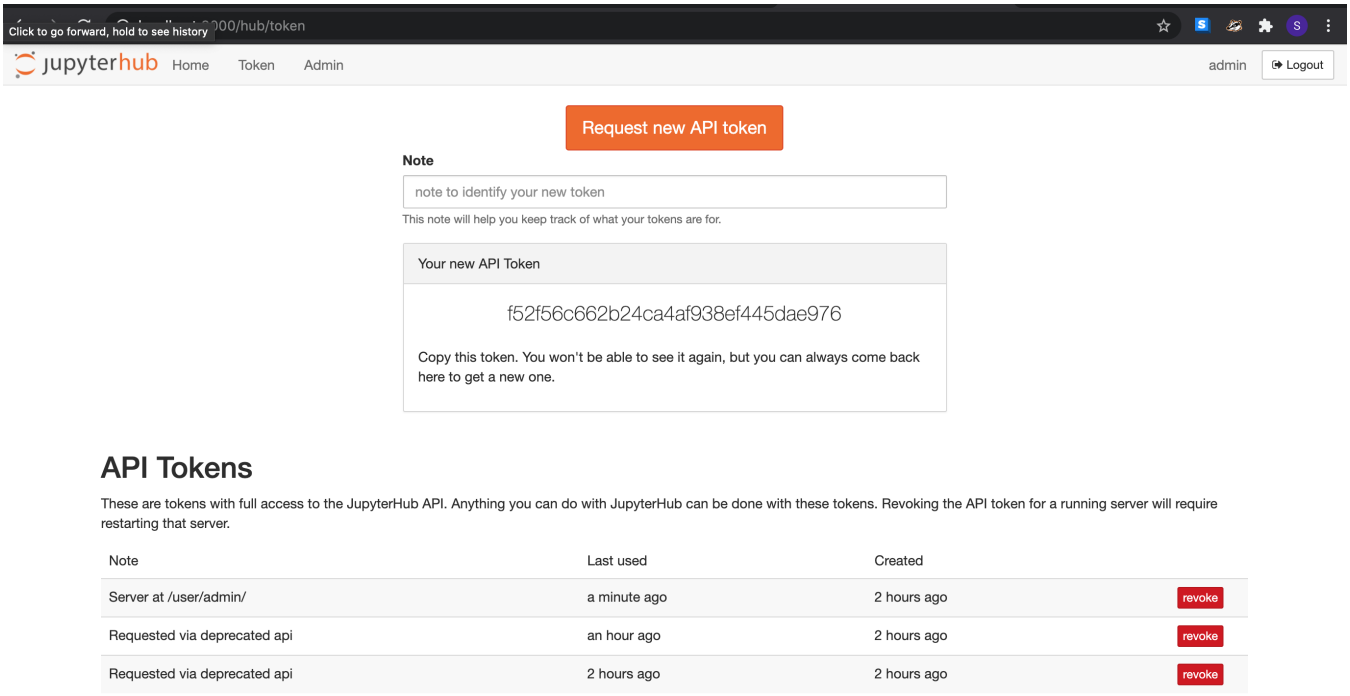
Uploading with the API

In addition to uploading data through the front end it is possible to add data programmatically through the REST API. All data that is added through the front end passes through this API so it is possible to perform any action that can be performed in the front end with the back end.

The REST API is available on port 5000 of the docker swarm "http://<server_address>:5000".

Access to the back end is secured with the same credentials as used by JupyterHub. To make calls programatically you will need a bearer token from Jupyter hub.

You can generate token for your user by visiting http://<server_address>:8000/hub/token" in your browser and clicking "Request Token"



All of the API endpoints are documented in 7.2 Back-End Implementation

For this example we will use the /table-fields endpoint to return a JSON object describing all of the tables and columns in the database as well as the /bulk-fields endpoint to write data into the database. The /bulk-fields endpoint takes a JSON list with the same format as the meta.json used in the previous upload example. As in that previous example the order we insert into the database matters with regard to satisfying key constraints.

We will call this data in Python but you could use any programming or scripting language capable of making HTTP requests.

In the header we are defining the Bearer token, you will need to replace the random string after Bearer with your own token from the previous step.

```
# Import required libraries
import requests
import json

# Define the API endpoints we are using local host in this example but your server name may vary
tf_endpoint = "http://localhost:5000/table-fields"
bf_endpoint = "http://localhost:5000/bulk-fields"

# Create an Authorization header with your Jupyter API token
header = {"Authorization": "Bearer f52f56c662b24ca4af938ef445dae976"}
```

We make a GET request to the table fields endpoint with our token and receive back a JSON object describing the data base

```
#Issue a GET request to the table fields API and store the JSON response (this has a type of Python Dictionary)
table_fields = requests.get(url=tf_endpoint, headers=header).json()
```

These JSON objects can be treated like any other Python iterable. We will print out a description of the database but you could use this information to automatically create a request to write data as shown in the next step.

```
# The response dictionary can be unpacked and iterated over
for table, fields in table_fields.items():
    print(f"Table: {table}")
    for field in fields:
        print(f"    Field name: {field['field']} Type: {field['type']}")
```

Here is a snippet of the results:

```
        Table: auditors
            Field name: first_name Type: varchar(45)
            Field name: last_name Type: varchar(45)
            Field name: employee_id Type: varchar(45)
        Table: audits
            Field name: audit_id Type: int
            Field name: audit_date Type: date
            Field name: revision_num Type: int
            Field name: facilities_facility_id Type: int
            Field name: versions_version Type: int
        Table: audits_has_auditors
            Field name: audits_audit_id Type: int
            Field name: audits_revision_num Type: int
            Field name: auditors_employee_id Type: varchar(45)
        Table: audits_has_reports
            Field name: audits_audit_id Type: int
            Field name: reports_report_id Type: int
        Table: audits_v
            Field name: audit_id Type: int
```

For the next example we will create our own JSON file and have it written into the database using a POST request to the /bulk-fields endpoint.

The data for this could come from any relevant source that you have available in python and be written to any table in the database.

This could enable large scale loading of historical data or automation of some task.

For this simple example we will call a free API on the internet that will return random names and write those into the auditors table.

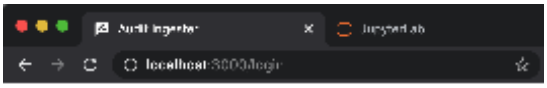
The results are stored in a dictionary object for each row and then those dictionary objects are added to a list. The format is identical to meta.json

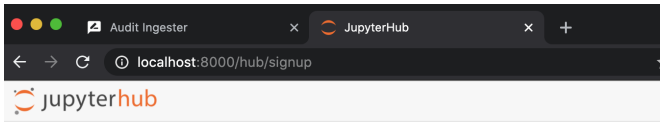
```
# Get Some data - in this case we are getting 50 random names from an API
response = requests.get("http://names.drycodes.com/50?nameOptions=girl_names").json()

# Create a list to hold each employee entry
auditors = list()
for i, name in enumerate(response):
    first, last = name.split('_')
    employee = {
        "table": "auditors",
        "fields": ["first_name","last_name","employee_id"],
        "values": [first, last ,str(i)]
        }
    auditors.append(employee)

# Turn the data structure to JSON string
data = json.dumps(auditors)

# We can submit this to the API via a POST request
response = requests.post(url=bf_endpoint, headers = header, json=data )
```

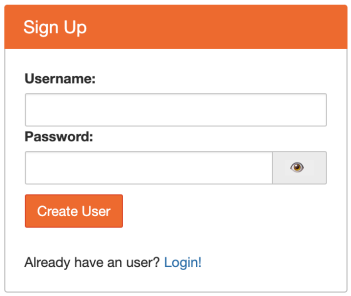
The system requires 3 files be provided for the audit: meta.json, facilty_stated.csv, and results.csv. Please see the guide in section 7.4 on how to prepare this data, or use the examples provided before beginning this guide.

Tutorial:

1. Navigate to the "Audit Ingester" web front end. This will be available on port 3000 of the server that is running the Platform Docker swarm such as "http://<server_address>:3000" . since I am running this on my local computer the address is http://localhost:3000. You will be asked to log in using the username and password that was created in Jupyter Labs, if you do not already have a username and password then please create one on the Jupyter hub login page which is available on the same server at port 8000 e.g. "http://<server_address>:8000".



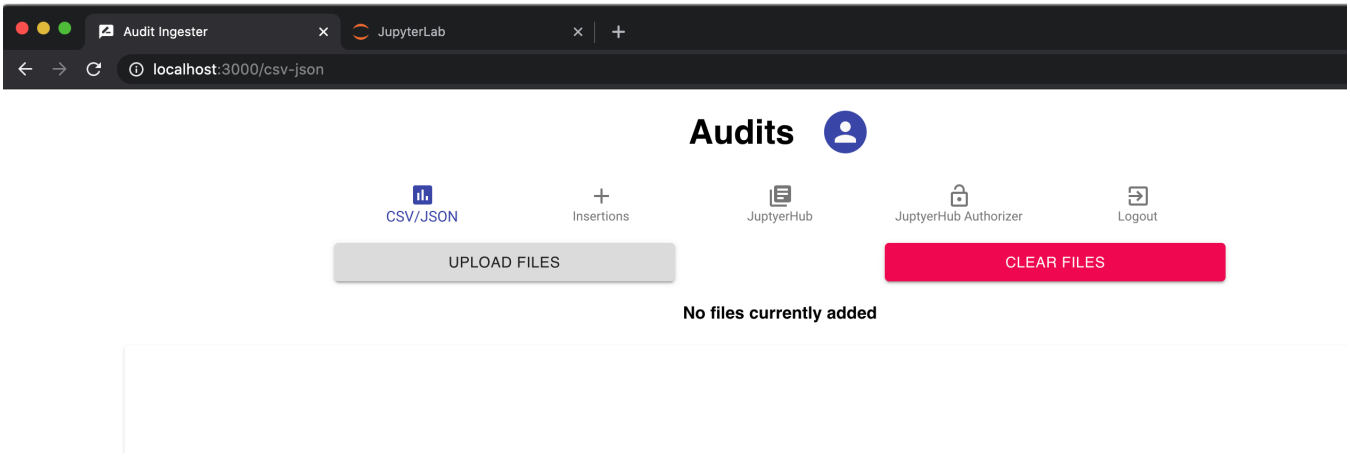




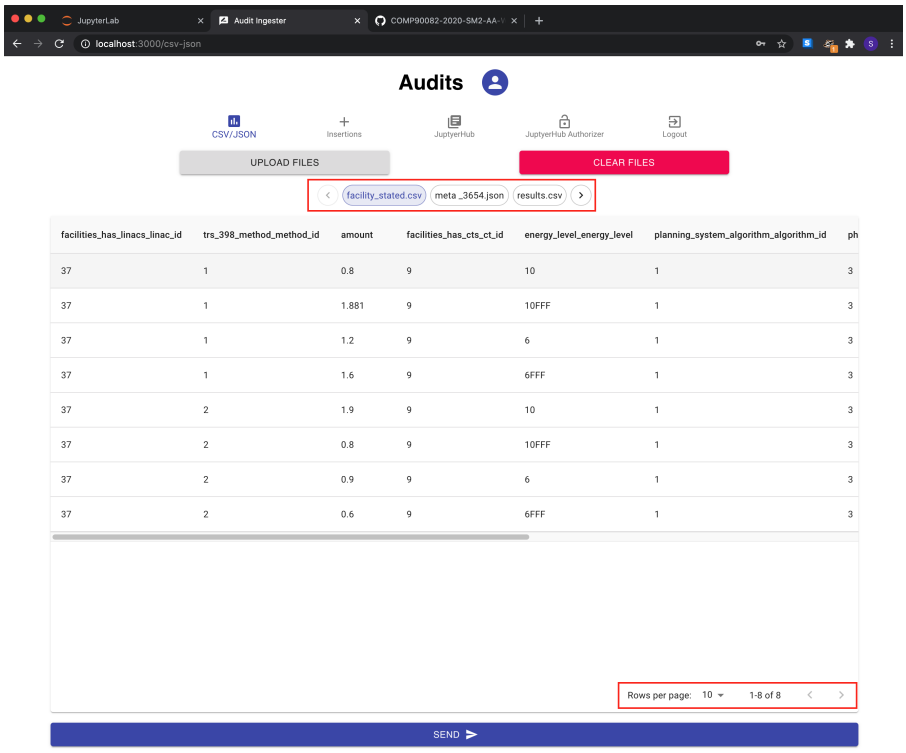
2. After logging in you should be redirected to the main page, ensuring that you have the CSV/JSON button selected click on "upload files". This button allows you to upload the files created in section 7.4. from your local computer.

You can upload one copy each of the files to be inserted into the database. As long as the foreign key constraints are satisfied, it is not necessary to upload all of the file at once. For example you could create the records for a specific audit by uploading the meta.json on one day, then providing the facility_stated.csv, and results.csv the next day after completing the audit.

For this tutorial our file contains data for one audit, but there is no limit to the number of audits that can be included within those three files, provided database key constraints are satisfied when inserting data.



3. For this example we will load will three files that describe the audit "3654" revision 1.

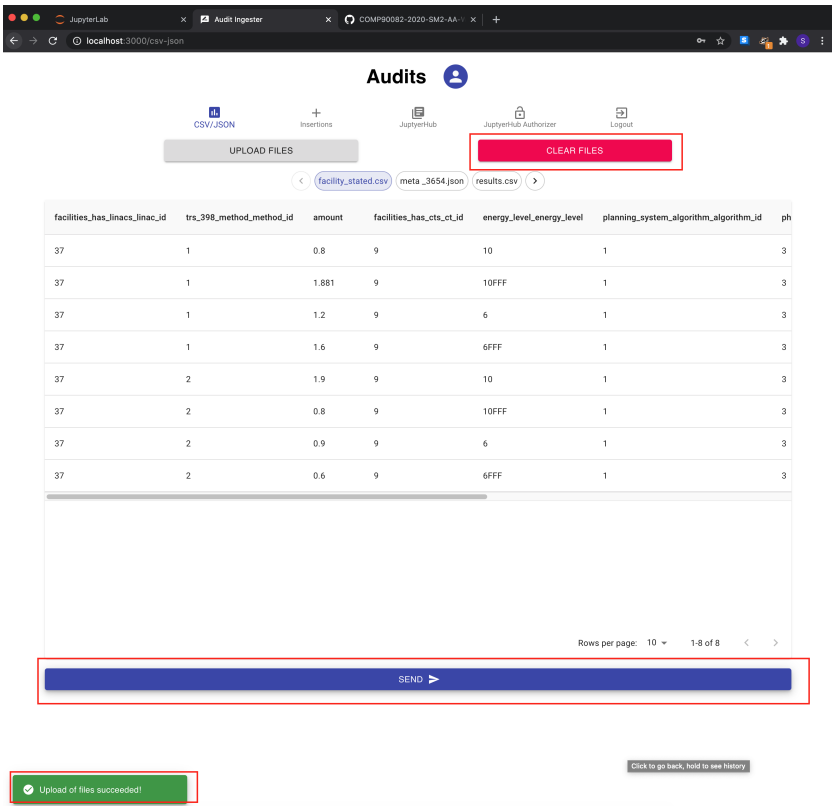


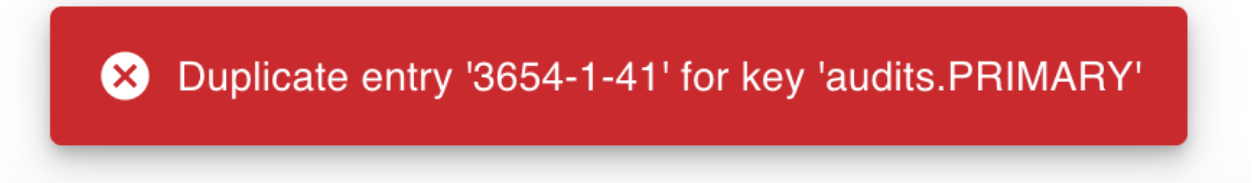
After uploading the files the system will read them and display the contents that it will insert into the database so that it can be verified before writing into the database.

The bar at the top (highlighted by the red box) shows the files that have been uploaded, you can scroll between each of the files to see a preview of the contents of each file. In the bottom right corner (also highlighted in red) are the pagination controls, for longer files it maybe necessary to page through them or scroll to see the entire contents.

4. If you are happy with the contents of the file then press the blue "send" bar at the bottom of the screen (highlighted in red) to write the data to the database. Upon writing to the database notification (highlighted in red) will slide in on the bottom left of the screen to let you know if the write was successful. If there are any errors then the notification will be red and contain the error message. The errors are SQL errors, passed up from the database and will most commonly be for key constraints, duplicates or incompatible values. If you receive an error you will need to correct it by clearing the files, correcting any errors in the file and re-uploading.

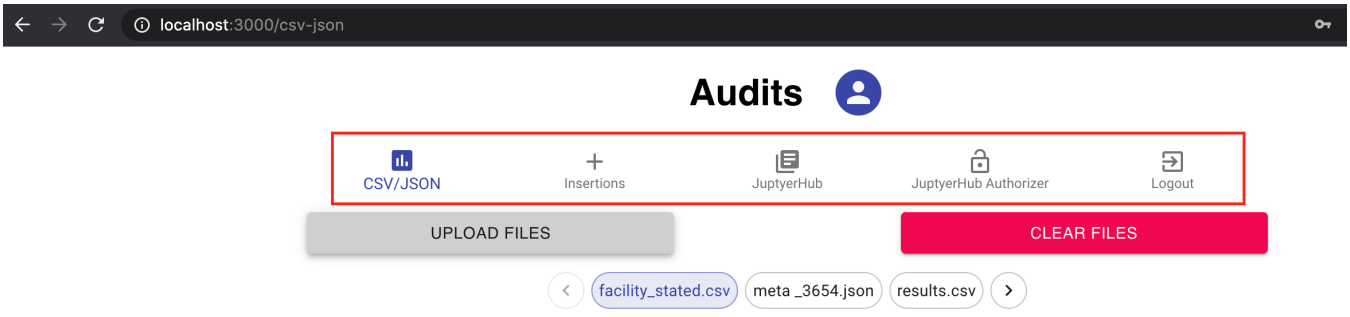
If you are not happy with the uploaded files and would like to cancel without writing into the database then you can do this by clicking the red "Clear Files" button.



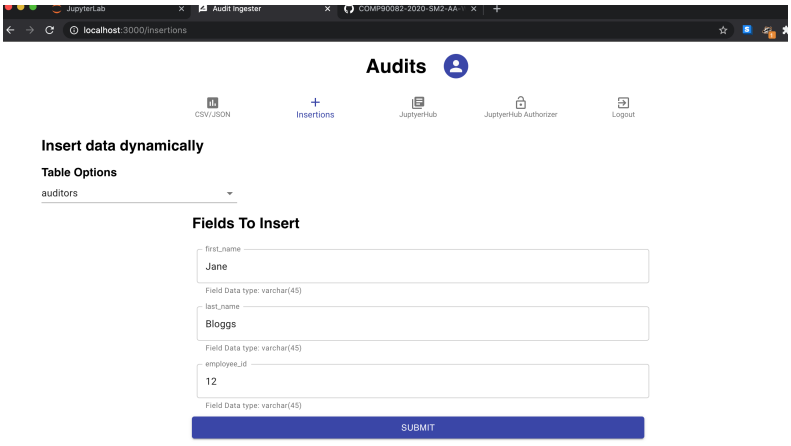


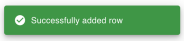
5. After writing the to the database and receiving your status information you have four options: Either clear the files and repeat the process to upload more data, use the inserts option to manually write more data to the database, click the JupyterHub button to analyse the data, or logout to end your session.

More information on performing analysis and generating reports with Jupyter can be found in Section 7.5 Jupyter Hub Tutorial.



6. For this tutorial we will use the insertions button (with the + sign) to add a new employee as an auditor and manually assign them to this audit, in addition to the 2 auditors that were specified in the meta.json file.





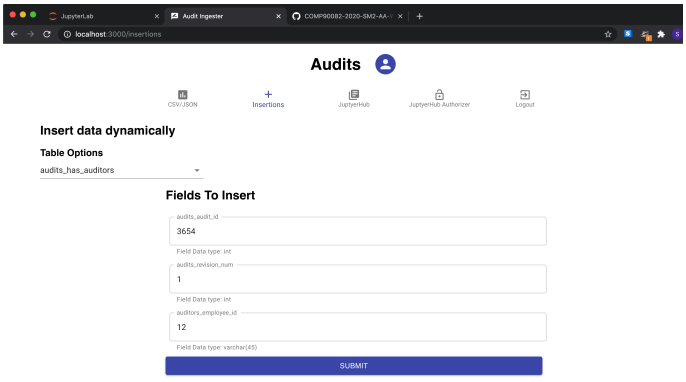
Information about ACDS employees who are performing audits is stored in the 'auditors' table. We can select the auditors table from the list of all available tables in the database from the drop down menu under "Insert data dynamically" - table options.

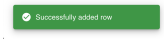
After selecting the table the page will update with all the fields (columns) in this table. This section, along with the table list are reflected dynamically from the database so should reflect any changes made to the underlying database automatically.

After entering the new employees information we can press the send button to write to the database and receive a notification to let us know whether or not the data was successfully inserted into the database.

7. Since many auditors will be working on many audits we will need to associate this auditor with the employee ID "12" to our audit ID "3654" with revision number "1".

This linking is done in the in the table"audits_has_auditors" which we can select in the same way form the "table options" drop down.





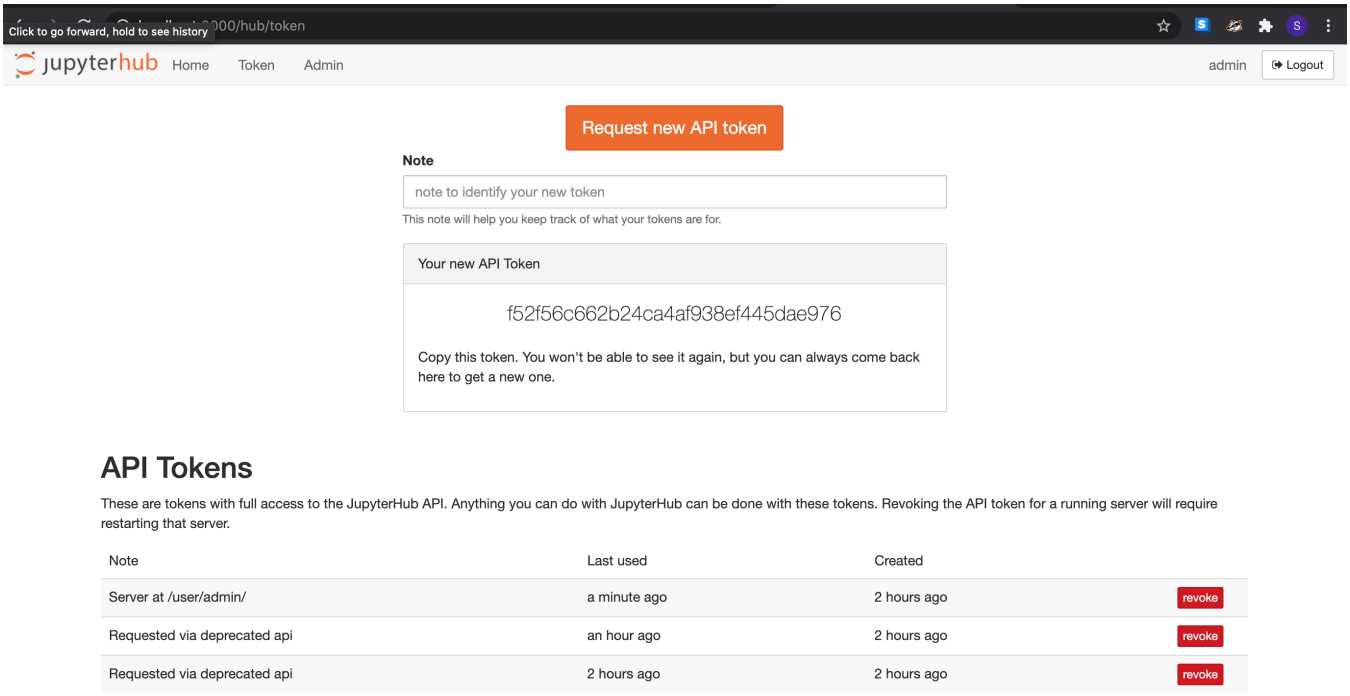
Uploading with the API

In addition to uploading data through the front end it is possible to add data programmatically through the REST API. All data that is added through the front end passes through this API so it is possible to perform any action that can be performed in the front end with the back end.

The REST API is available on port 5000 of the docker swarm "http://<server_address>:5000".

Access to the back end is secured with the same credentials as used by JupyterHub. To make calls programatically you will need a bearer token from Jupyter hub.

You can generate token for your user by visiting http://<server_address>:8000/hub/token" in your browser and clicking "Request Token"



All of the API endpoints are documented in 7.2 Back-End Implementation

For this example we will use the /table-fields endpoint to return a JSON object describing all of the tables and columns in the database as well as the /bulk-fields endpoint to write data into the database. The /bulk-fields endpoint takes a JSON list with the same format as the meta.json used in the previous upload example. As in that previous example the order we insert into the database matters with regard to satisfying key constraints.

We will call this data in Python but you could use any programming or scripting language capable of making HTTP requests.

In the header we are defining the Bearer token, you will need to replace the random string after Bearer with your own token from the previous step.

```
# Import required libraries
import requests
import json

# Define the API endpoints we are using local host in this example but your server name may vary
tf_endpoint = "http://localhost:5000/table-fields"
bf_endpoint = "http://localhost:5000/bulk-fields"

# Create an Authorization header with your Jupyter API token
header = {"Authorization": "Bearer f52f56c662b24ca4af938ef445dae976"}
```

We make a GET request to the table fields endpoint with our token and receive back a JSON object describing the data base

```
#Issue a GET request to the table fields API and store the JSON response (this has a type of Python Dictionary)
table_fields = requests.get(url=tf_endpoint, headers=header).json()
```

These JSON objects can be treated like any other Python iterable. We will print out a description of the database but you could use this information to automatically create a request to write data as shown in the next step.

```
# The response dictionary can be unpacked and iterated over
for table, fields in table_fields.items():
    print(f"Table: {table}")
    for field in fields:
        print(f"    Field name: {field['field']} Type: {field['type']}")
```

Here is a snippet of the results:

```
        Table: auditors
            Field name: first_name Type: varchar(45)
            Field name: last_name Type: varchar(45)
            Field name: employee_id Type: varchar(45)
        Table: audits
            Field name: audit_id Type: int
            Field name: audit_date Type: date
            Field name: revision_num Type: int
            Field name: facilities_facility_id Type: int
            Field name: versions_version Type: int
        Table: audits_has_auditors
            Field name: audits_audit_id Type: int
            Field name: audits_revision_num Type: int
            Field name: auditors_employee_id Type: varchar(45)
        Table: audits_has_reports
            Field name: audits_audit_id Type: int
            Field name: reports_report_id Type: int
        Table: audits_v
            Field name: audit_id Type: int
```

For the next example we will create our own JSON file and have it written into the database using a POST request to the /bulk-fields endpoint.

The data for this could come from any relevant source that you have available in python and be written to any table in the database.

This could enable large scale loading of historical data or automation of some task.

For this simple example we will call a free API on the internet that will return random names and write those into the auditors table.

The results are stored in a dictionary object for each row and then those dictionary objects are added to a list. The format is identical to meta.json

```
# Get Some data - in this case we are getting 50 random names from an API
response = requests.get("http://names.drycodes.com/50?nameOptions=girl_names").json()

# Create a list to hold each employee entry
auditors = list()
for i, name in enumerate(response):
    first, last = name.split('_')
    employee = {
        "table": "auditors",
        "fields": ["first_name","last_name","employee_id"],
        "values": [first, last ,str(i)]
        }
    auditors.append(employee)

# Turn the data structure to JSON string
data = json.dumps(auditors)

# We can submit this to the API via a POST request
response = requests.post(url=bf_endpoint, headers = header, json=data )
```

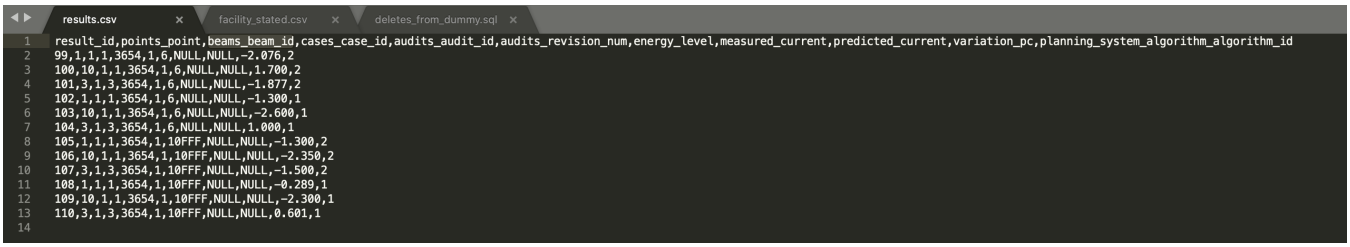
6.2 Data Preparation Tutorial

Data Preparation

Currently the system requires a fair amount of data preparation to be inserted into the system. This is an area where there is scope for further improvement to build more logic into the system to handle the ingestion process more automatically and require less preprocessing on the part of the person loading the data in.

The system requires 3 files for a complete audit although it is possible to insert any or all of this data into the database through the "Insertions" page on the web app, (or alternatively through SQL by connecting directly to the data base. To insert data is important to understand the database structure, data types, and dependancies. These are documented in further detail in section 6 Database.

results.csv



This is a comma delimited CSV file that contains the post audit results that have been measured through a Phantom. The file must contain 'results' in the file name and have the extension .csv "audit_3654_results.csv" for example would be a valid file name.

Please Note:

The first row of the file must contain the column names as they appear in the database.

For example:

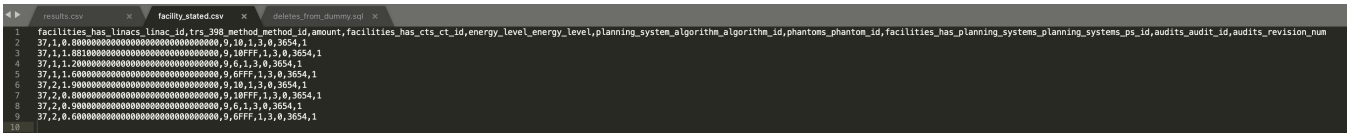
result_id,points_point,beams_beam_id,cases_case_id,audits_audit_id,audits_revision_num,energy_level,measured_current,predicted_current, variation_pc,planning_system_algorithm_algorithm_id

The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)

Any NULL values must appear exactly as NULL, entering Null, null, or leaving a blank will cause an error.

1. result_id = this is an integer value unique to that row. These can either be randomly generated or sequential provided they do not already appear in the results database table.
2. points_point = integer value for the point in the phantom that the result relates to. Value must appear in SELECT * FROM aa_audit.points;
3. beams_beam_id = integer value for the beam that the result relates to. Value must appear in SELECT * FROM aa_audit.beams;
4. cases_case_id = integer value for the case that the result relates to. Value must appear in SELECT * FROM aa_audit.cases;
5. audits_audit_id = integer value for the audit ID that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)
6. audits_revision_num = integer value for the audit revision number that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file.
7. energy_level = integer value for the energy level that the result relates to. Value must appear in SELECT * FROM aa_audit.energy_level;
8. measured_current = decimal value or NULL, this is an optional observation taken from the Phantom.
9. predicted_current = decimal value or NULL, this is an optional prediction of the measured current taken from the Phantom based on the facility stated information . This value would need to be calculated before insertion.
10. variation_pc = positive or negative decimal value representing the percentage variation between the predicted current and measured current from the Phantom. This value would need to be calculated before insertion.
11. planning_system_algorithm_algorithm_id = integer value representing the ID of the algorithm used by the planning system for this record. Value must appear in SELECT * FROM aa_audit.planning_system_algorithm;

facility_stated.csv



This is a comma delimited CSV file that contains the pre audit results that have been provided by a facility on the day of an audit. The file must contain 'facility_stated' in the file name and have the extension .csv "audit_3654_facility_stated.csv" for example would be a valid file name.

Please Note:

The first row of the file must contain the column names as they appear in the database.

For example:

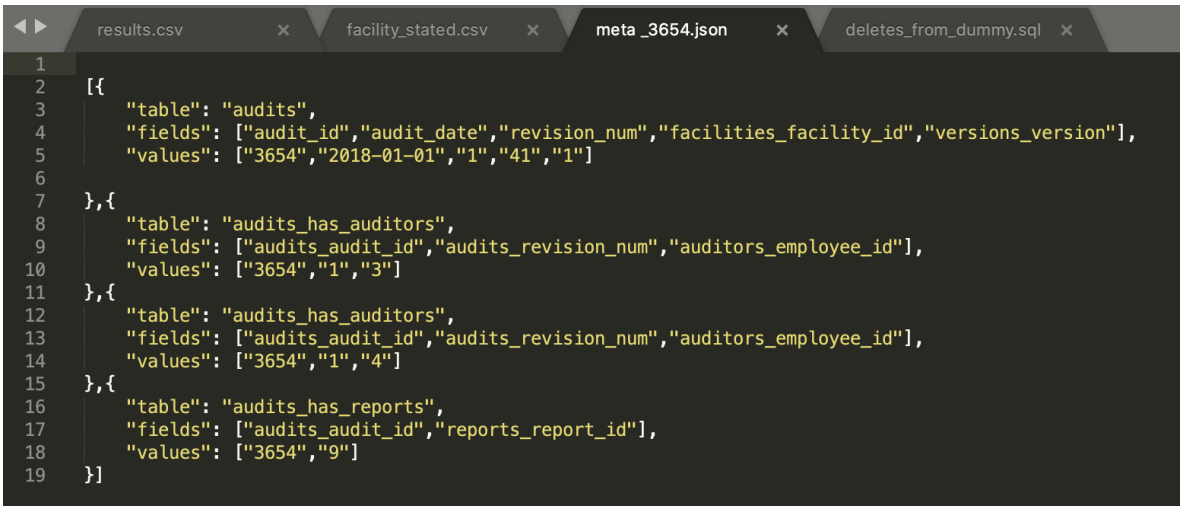
facilities_has_linacs_linac_id,trs_398_method_method_id,amount,facilities_has_cts_ct_id,energy_level_energy_level,
planning_system_algorithm_algorithm_id,phantoms_phantom_id,facilities_has_planning_systems_planning_systems_ps_id,audits_audit_id,
audits_revision_num

The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)

Any NULL values must appear exactly as NULL, entering Null, null, or leaving a blank will cause an error.

1. facilities_has_linacs_linac_id = integer value for the linac id corresponding to the linac, belonging to a specific facility being audited. Value must appear in SELECT * FROM aa_audit.facilities_has_linacs;
2. trs_398_method_method_id = integer value for the trs 398 method used by the facility that the amount value relates to. Value must appear in SELECT * FROM aa_audit.trs_398_method;
3. amount = decimal value representing the value the facility has provided for that trs 398 method and energy level.
4. facilities_has_cts_ct_id = integer value for the CT machine id used in the planning process and belonging to a specific facility being audited. Value must appear in SELECT * FROM aa_audit.facilities_has_cts;
5. energy_level = integer value for the energy level that the result relates to. Value must appear in SELECT * FROM aa_audit.energy_level;
6. planning_system_algorithm_algorithm_id = integer value representing the ID of the algorithm used by the planning system for this record. Value must appear in SELECT * FROM aa_audit.planning_system_algorithm;
7. phantoms_phantom_id = integer value for the phantom id corresponding to the phantom being used for the audit. Value must appear in SELECT * FROM aa_audit.phantoms;
8. facilities_has_planning_systems_planning_systems_ps_id = integer value representing the ID of the planning system used by the facility this audit. Value must appear in SELECT * FROM aa_audit.planning_systems;
9. audits_audit_id = integer value for the audit ID that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)
10. audits_revision_num = integer value for the audit revision number that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file.

meta.json



The meta.json is JSON file allowing you to specify all other metadata to be inserted into any other table in the database. When loaded with a group files this file will be executed before results.csv and facility_stated.csv. The object in this file is one list of database inserts and the system will insert the data in the order it appears in this file so it is important that the order respects the foreign key constraints of the database.

The table names and field names need to match the database exactly. All values should be entered in quotes like "354.59" regardless of data type. The system will cast the value to the correct type when inserted.

It is only necessary to insert records for the tables you want to insert data into. Also it is only necessary to specify the fields and values that you have data to enter. If a column in the database supports NULLS and you would like that value to be NULL then simply leave that column name and value out of the fields and values lists and it will be inserted as NULL into the database.

You can insert multiple rows into the same table by having repeating entries like the "audits_has_aditors" entry in the example shown.

Example Files

If using the below example files it may be necessary to first remove them form the database if this audit record already exists.

The below SQL snippet will delete the below data from the database (note that to delete records they need to be removed in the reverse order from inserting so that foreign key constraints are satisfied).

Remove audit 3654

```
DELETE FROM aa_audit.audits_has_auditors WHERE audits_audit_id = 3654;
DELETE FROM aa_audit.audits_has_reports where audits_audit_id = 3654;

DELETE FROM aa_audit.facility_stated WHERE audits_audit_id = 3654;
DELETE FROM aa_audit.results WHERE audits_audit_id = 3654;

DELETE FROM aa_audit.audits WHERE audit_id = 3654;
```

results.csv

```
result_id,points_point,beams_beam_id,cases_case_id,audits_audit_id,audits_revision_num,energy_level,
measured_current,predicted_current,variation_pc,planning_system_algorithm_algorithm_id
99,1,1,1,3654,1,6,NULL,NULL,-2.076,2
100,10,1,1,3654,1,6,NULL,NULL,1.700,2
101,3,1,3,3654,1,6,NULL,NULL,-1.877,2
102,1,1,1,3654,1,6,NULL,NULL,-1.300,1
103,10,1,1,3654,1,6,NULL,NULL,-2.600,1
104,3,1,3,3654,1,6,NULL,NULL,1.000,1
105,1,1,1,3654,1,10FFF,NULL,NULL,-1.300,2
106,10,1,1,3654,1,10FFF,NULL,NULL,-2.350,2
107,3,1,3,3654,1,10FFF,NULL,NULL,-1.500,2
108,1,1,1,3654,1,10FFF,NULL,NULL,-0.289,1
109,10,1,1,3654,1,10FFF,NULL,NULL,-2.300,1
110,3,1,3,3654,1,10FFF,NULL,NULL,0.601,1
```

facility_stated.csv

```
facilities_has_linacs_linac_id,trs_398_method_method_id,amount,facilities_has_cts_ct_id,
energy_level_energy_level,planning_system_algorithm_algorithm_id,phantoms_phantom_id,
facilities_has_planning_systems_planning_systems_ps_id,audits_audit_id,audits_revision_num
37,1,0.8000000000000000000000000000000,9,10,1,3,0,3654,1
37,1,1.8810000000000000000000000000000,9,10FFF,1,3,0,3654,1
37,1,1.2000000000000000000000000000000,9,6,1,3,0,3654,1
37,1,1.6000000000000000000000000000000,9,6FFF,1,3,0,3654,1
37,2,1.9000000000000000000000000000000,9,10,1,3,0,3654,1
37,2,0.8000000000000000000000000000000,9,10FFF,1,3,0,3654,1
37,2,0.9000000000000000000000000000000,9,6,1,3,0,3654,1
37,2,0.6000000000000000000000000000000,9,6FFF,1,3,0,3654,1
```

meta.json

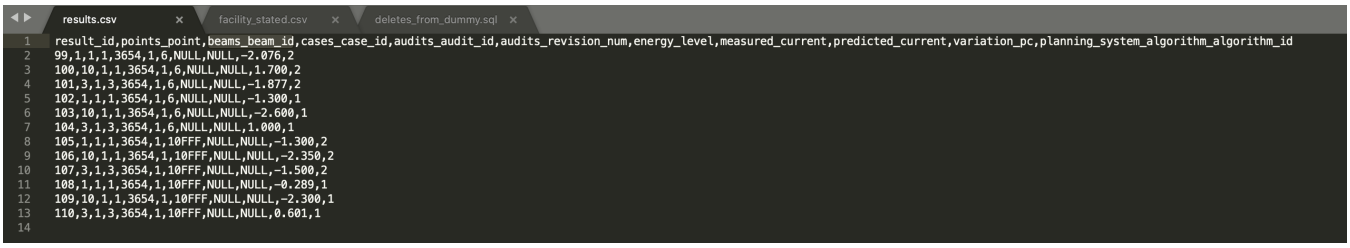
```
[{
    "table": "audits",
    "fields": ["audit_id","audit_date","revision_num","facilities_facility_id","versions_version"],
    "values": ["3654","2018-01-01","1","41","1"]

},{
    "table": "audits_has_auditors",
    "fields": ["audits_audit_id","audits_revision_num","auditors_employee_id"],
    "values": ["3654","1","3"]
},{
    "table": "audits_has_auditors",
    "fields": ["audits_audit_id","audits_revision_num","auditors_employee_id"],
    "values": ["3654","1","4"]
},{
    "table": "audits_has_reports",
    "fields": ["audits_audit_id","reports_report_id"],
    "values": ["3654","9"]
}]
```

Data Preparation

Currently the system requires a fair amount of data preparation to be inserted into the system. This is an area where there is scope for further improvement to build more logic into the system to handle the ingestion process more automatically and require less preprocessing on the part of the person loading the data in.

The system requires 3 files for a complete audit although it is possible to insert any or all of this data into the database through the "Insertions" page on the web app, (or alternatively through SQL by connecting directly to the data base. To insert data is important to understand the database structure, data types, and dependancies. These are documented in further detail in section 6 Database.

results.csv



This is a comma delimited CSV file that contains the post audit results that have been measured through a Phantom. The file must contain 'results' in the file name and have the extension .csv "audit_3654_results.csv" for example would be a valid file name.

Please Note:

The first row of the file must contain the column names as they appear in the database.

For example:

result_id,points_point,beams_beam_id,cases_case_id,audits_audit_id,audits_revision_num,energy_level,measured_current,predicted_current, variation_pc,planning_system_algorithm_algorithm_id

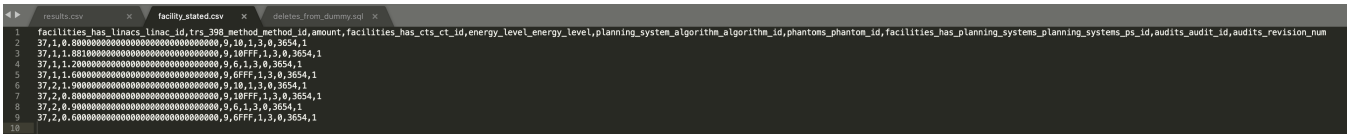
The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)

Any NULL values must appear exactly as NULL, entering Null, null, or leaving a blank will cause an error.

1. result_id = this is an integer value unique to that row. These can either be randomly generated or sequential provided they do not already appear in the results database table.
2. points_point = integer value for the point in the phantom that the result relates to. Value must appear in SELECT * FROM aa_audit.points;
3. beams_beam_id = integer value for the beam that the result relates to. Value must appear in SELECT * FROM aa_audit.beams;
4. cases_case_id = integer value for the case that the result relates to. Value must appear in SELECT * FROM aa_audit.cases;
5. audits_audit_id = integer value for the audit ID that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)
6. audits_revision_num = integer value for the audit revision number that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file.
7. energy_level = integer value for the energy level that the result relates to. Value must appear in SELECT * FROM aa_audit.energy_level;
8. measured_current = decimal value or NULL, this is an optional observation taken from the Phantom.
9. predicted_current = decimal value or NULL, this is an optional prediction of the measured current taken from the Phantom based on the facility stated information . This value would need to be calculated before insertion.
10. variation_pc = positive or negative decimal value representing the percentage variation between the predicted current and measured current from the Phantom. This value would need to be calculated before insertion.

11. planning_system_algorithm_algorithm_id = integer value representing the ID of the algorithm used by the planning system for this record. Value must appear in SELECT * FROM aa_audit.planning_system_algorithm;

facility_stated.csv



This is a comma delimited CSV file that contains the pre audit results that have been provided by a facility on the day of an audit. The file must contain 'facility_stated' in the file name and have the extension .csv "audit_3654_facility_stated.csv" for example would be a valid file name.

Please Note:

The first row of the file must contain the column names as they appear in the database.

For example:

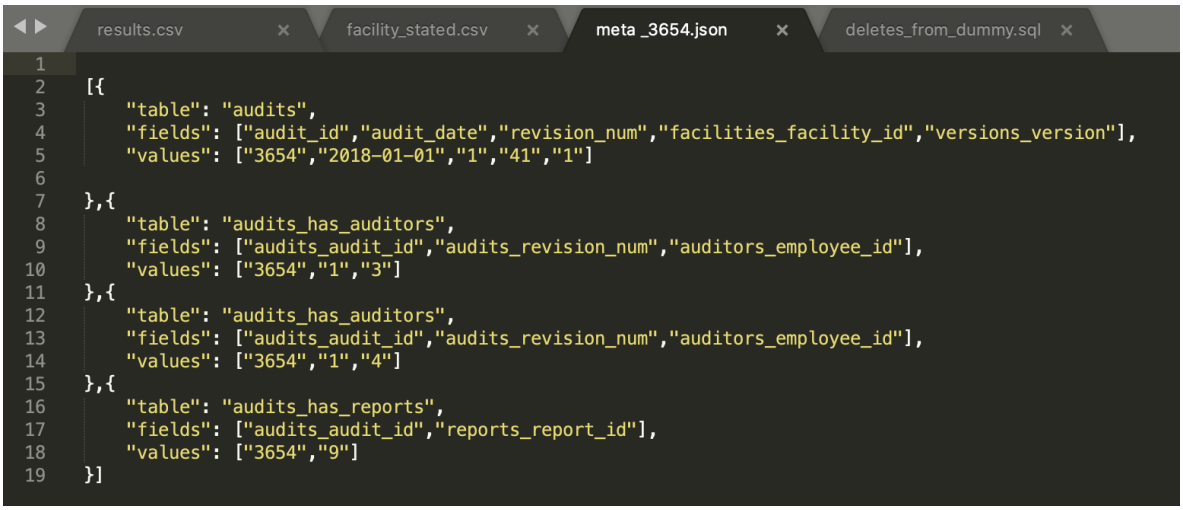
facilities_has_linacs_linac_id,trs_398_method_method_id,amount,facilities_has_cts_ct_id,energy_level_energy_level,
planning_system_algorithm_algorithm_id,phantoms_phantom_id,facilities_has_planning_systems_planning_systems_ps_id,audits_audit_id,
audits_revision_num

The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)

Any NULL values must appear exactly as NULL, entering Null, null, or leaving a blank will cause an error.

1. facilities_has_linacs_linac_id = integer value for the linac id corresponding to the linac, belonging to a specific facility being audited. Value must appear in SELECT * FROM aa_audit.facilities_has_linacs;
2. trs_398_method_method_id = integer value for the trs 398 method used by the facility that the amount value relates to. Value must appear in SELECT * FROM aa_audit.trs_398_method;
3. amount = decimal value representing the value the facility has provided for that trs 398 method and energy level.
4. facilities_has_cts_ct_id = integer value for the CT machine id used in the planning process and belonging to a specific facility being audited. Value must appear in SELECT * FROM aa_audit.facilities_has_cts;
5. energy_level = integer value for the energy level that the result relates to. Value must appear in SELECT * FROM aa_audit.energy_level;
6. planning_system_algorithm_algorithm_id = integer value representing the ID of the algorithm used by the planning system for this record. Value must appear in SELECT * FROM aa_audit.planning_system_algorithm;
7. phantoms_phantom_id = integer value for the phantom id corresponding to the phantom being used for the audit. Value must appear in SELECT * FROM aa_audit.phantoms;
8. facilities_has_planning_systems_planning_systems_ps_id = integer value representing the ID of the planning system used by the facility this audit. Value must appear in SELECT * FROM aa_audit.planning_systems;
9. audits_audit_id = integer value for the audit ID that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file)
10. audits_revision_num = integer value for the audit revision number that result relates to. The Audit ID and revision number must already have been entered in the audits table (or exist in the meta.json file.

meta.json



The meta.json is JSON file allowing you to specify all other metadata to be inserted into any other table in the database. When loaded with a group files this file will be executed before results.csv and facility_stated.csv. The object in this file is one list of database inserts and the system will insert the data in the order it appears in this file so it is important that the order respects the foreign key constraints of the database.

The table names and field names need to match the database exactly. All values should be entered in quotes like "354.59" regardless of data type. The system will cast the value to the correct type when inserted.

It is only necessary to insert records for the tables you want to insert data into. Also it is only necessary to specify the fields and values that you have data to enter. If a column in the database supports NULLS and you would like that value to be NULL then simply leave that column name and value out of the fields and values lists and it will be inserted as NULL into the database.

You can insert multiple rows into the same table by having repeating entries like the "audits_has_aditors" entry in the example shown.

Example Files

If using the below example files it may be necessary to first remove them form the database if this audit record already exists.

The below SQL snippet will delete the below data from the database (note that to delete records they need to be removed in the reverse order from inserting so that foreign key constraints are satisfied).

Remove audit 3654

```
DELETE FROM aa_audit.audits_has_auditors WHERE audits_audit_id = 3654;
DELETE FROM aa_audit.audits_has_reports where audits_audit_id = 3654;

DELETE FROM aa_audit.facility_stated WHERE audits_audit_id = 3654;
DELETE FROM aa_audit.results WHERE audits_audit_id = 3654;

DELETE FROM aa_audit.audits WHERE audit_id = 3654;
```

results.csv

```
result_id,points_point,beams_beam_id,cases_case_id,audits_audit_id,audits_revision_num,energy_level,
measured_current,predicted_current,variation_pc,planning_system_algorithm_algorithm_id
99,1,1,1,3654,1,6,NULL,NULL,-2.076,2
100,10,1,1,3654,1,6,NULL,NULL,1.700,2
101,3,1,3,3654,1,6,NULL,NULL,-1.877,2
102,1,1,1,3654,1,6,NULL,NULL,-1.300,1
103,10,1,1,3654,1,6,NULL,NULL,-2.600,1
104,3,1,3,3654,1,6,NULL,NULL,1.000,1
105,1,1,1,3654,1,10FFF,NULL,NULL,-1.300,2
106,10,1,1,3654,1,10FFF,NULL,NULL,-2.350,2
107,3,1,3,3654,1,10FFF,NULL,NULL,-1.500,2
108,1,1,1,3654,1,10FFF,NULL,NULL,-0.289,1
109,10,1,1,3654,1,10FFF,NULL,NULL,-2.300,1
110,3,1,3,3654,1,10FFF,NULL,NULL,0.601,1
```

facility_stated.csv

```
facilities_has_linacs_linac_id,trs_398_method_method_id,amount,facilities_has_cts_ct_id,
energy_level_energy_level,planning_system_algorithm_algorithm_id,phantoms_phantom_id,
facilities_has_planning_systems_planning_systems_ps_id,audits_audit_id,audits_revision_num
37,1,0.8000000000000000000000000000000,9,10,1,3,0,3654,1
37,1,1.8810000000000000000000000000000,9,10FFF,1,3,0,3654,1
37,1,1.2000000000000000000000000000000,9,6,1,3,0,3654,1
37,1,1.6000000000000000000000000000000,9,6FFF,1,3,0,3654,1
37,2,1.9000000000000000000000000000000,9,10,1,3,0,3654,1
37,2,0.8000000000000000000000000000000,9,10FFF,1,3,0,3654,1
37,2,0.9000000000000000000000000000000,9,6,1,3,0,3654,1
37,2,0.6000000000000000000000000000000,9,6FFF,1,3,0,3654,1
```

meta.json


```
[{
    "table": "audits",
    "fields": ["audit_id","audit_date","revision_num","facilities_facility_id","versions_version"],
    "values": ["3654","2018-01-01","1","41","1"]

},{
    "table": "audits_has_auditors",
    "fields": ["audits_audit_id","audits_revision_num","auditors_employee_id"],
    "values": ["3654","1","3"]
},{
    "table": "audits_has_auditors",
    "fields": ["audits_audit_id","audits_revision_num","auditors_employee_id"],
    "values": ["3654","1","4"]
},{
    "table": "audits_has_reports",
    "fields": ["audits_audit_id","reports_report_id"],
    "values": ["3654","9"]
}]
```

6.3 Jupyter Tutorial

Overview:

Jupyter hub provides a centralised, collaborative platform for teams to perform analysis on data in the system, and publish reports. It is also responsible for managing and authenticating users throughout the system.

JupyterHub will run on the the server hosting the platform inside a Docker container, spawning a new container for each user that connects. This ensures consistency as all users are always looking at the same data and using the same code libraries (including custom code libraries that have been written by ACDS) , no matter where they are working from. It also means no software is required on the users computer or device except for a web browser (and the ability to connect to the server hosted by ARPANSA).

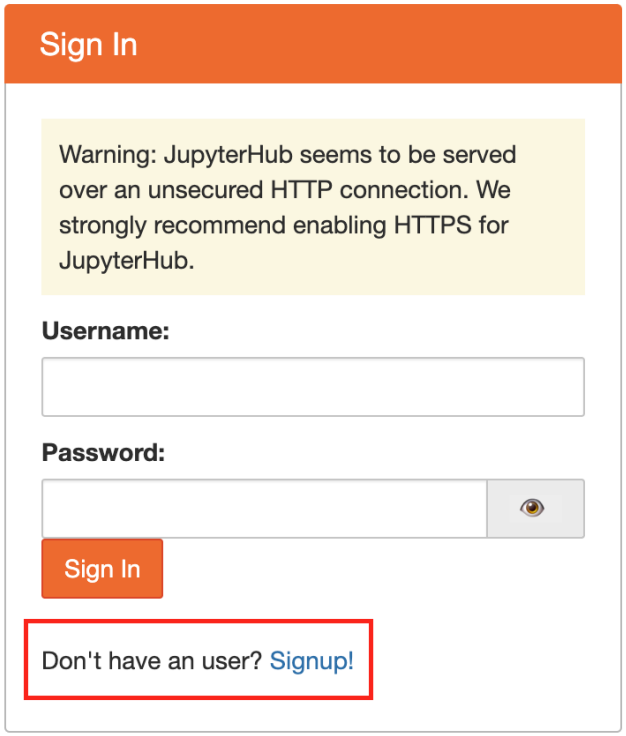
https://jupyter.org/hub

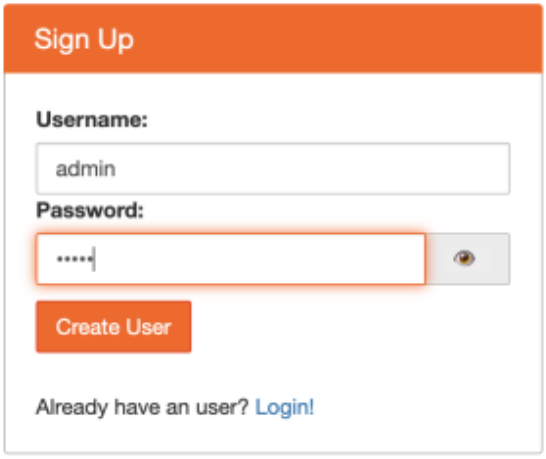
Creating Users:

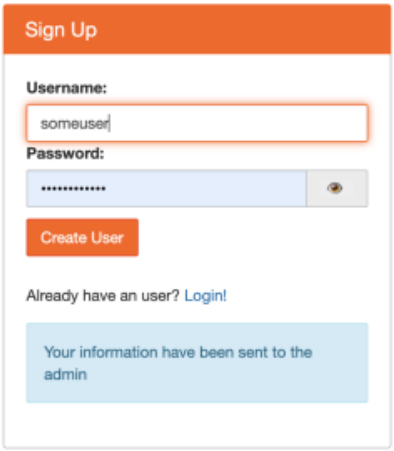
Jupyter Hub is responsible for managing user authentication for the web applications and APIs in the system (everything except the database users).

The first time the Platform is started it is necessary to create an admin user, and other users for whoever else will want to use the system.

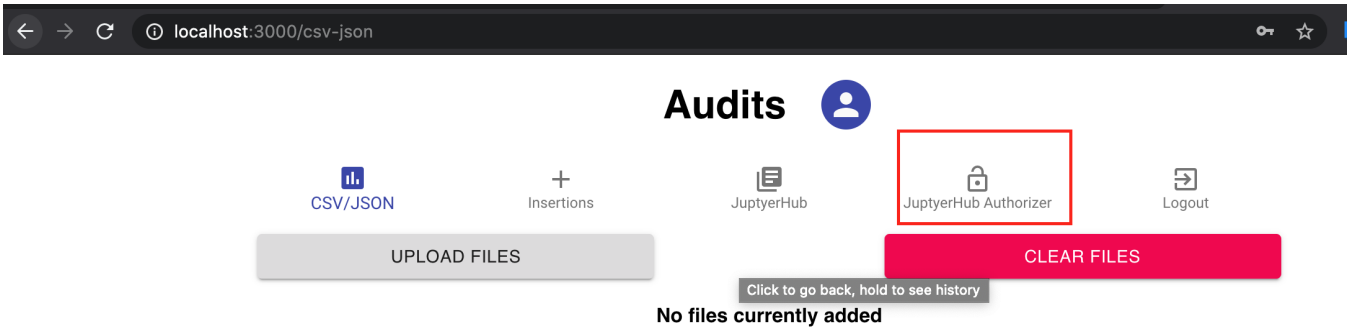
When the platform is started JupyterHub will be available at port 8000 on the server that is running the platform at http://<server_name>:8000







Click on the Signup button and enter the user name 'admin' and a password of your choosing. This admin user will not need to be authenticated the first time it is created and can be used to authorise other normal users in the system. Other normal users can go to this signup page and request account on the system



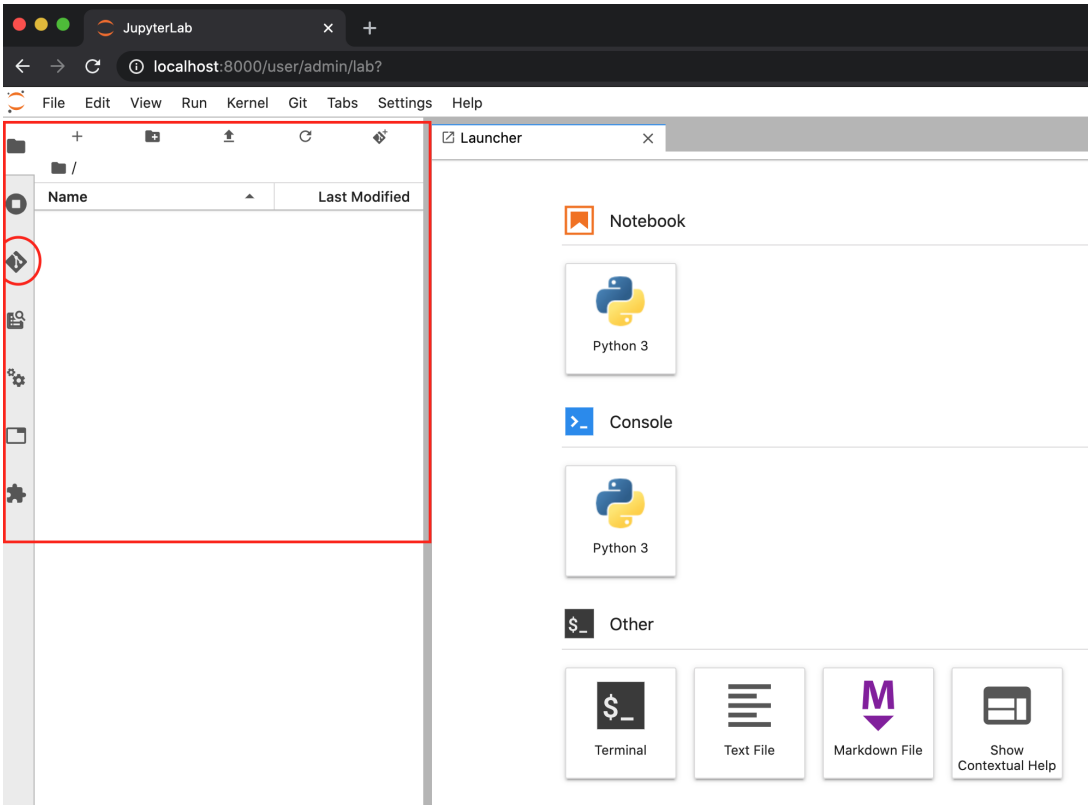
the admin user can then authenticate the other user by going to the JupyterHub Authorizer page. There is a link to this page (visible only to admin) in the audit ingestion page or through http://<server_name>:8000/hub/authorize

From this page admin can authorise other users for the system.

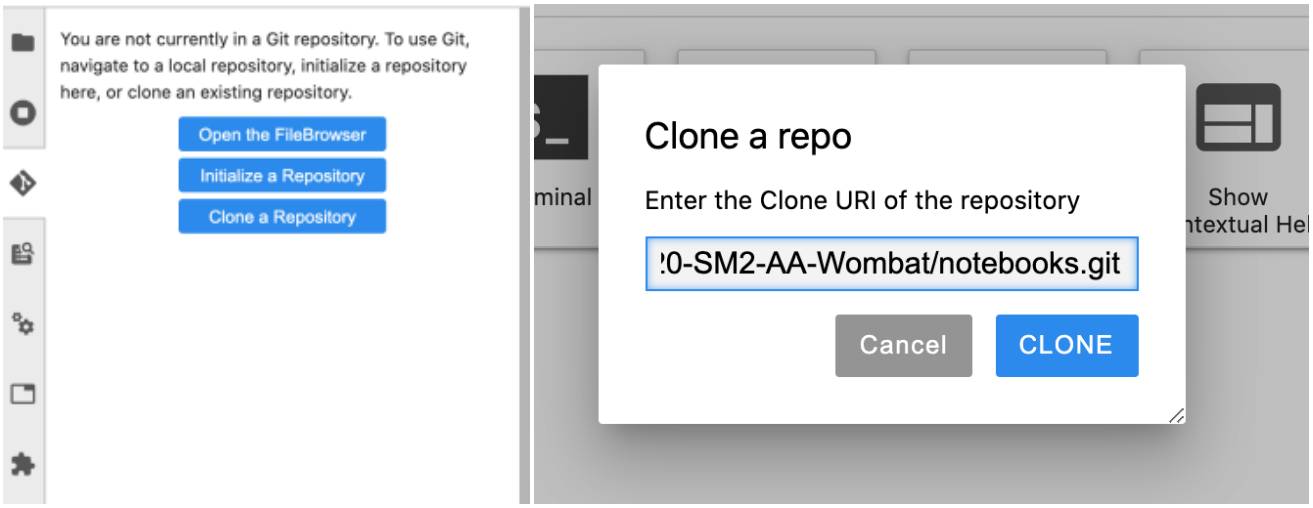
Jupyter Lab

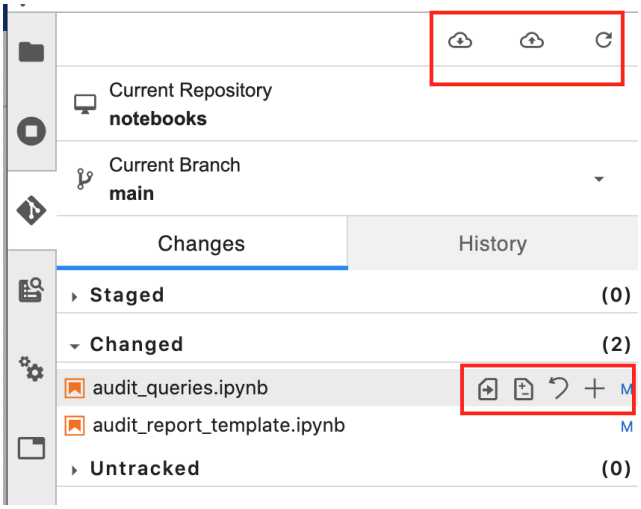
Git Extension:

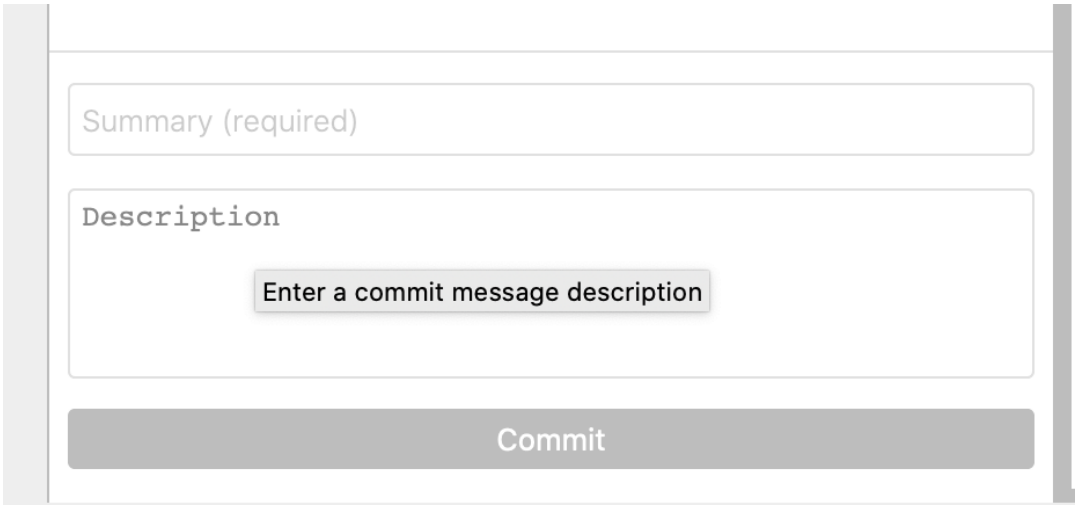
After logging into Jupyter Lab the user will be presented with the below screen. It is important to note that Jupyter runs in ephemeral containers on a remote server. To load and save Jupyter notebooks you will need to to use the Git extension. This connects to a Git repository so that changes to notebooks can be version controlled. The project comes with a notebooks repo already (https://github.com/COMP90082-2020-SM2-AA-Wombat/notebooks.git) but you can connect to any Git repository that you like. To access Git through the Jupyter notebook please use the Git extension panel on the left of the screen, and start by clicking the Git Icon (circled in red).



We will use the clone a repository option, and enter the repo URL https://github.com/COMP90082-2020-SM2-AA-Wombat/notebooks.git. Logging in with a username and password when prompted (other options like tokens are possible).

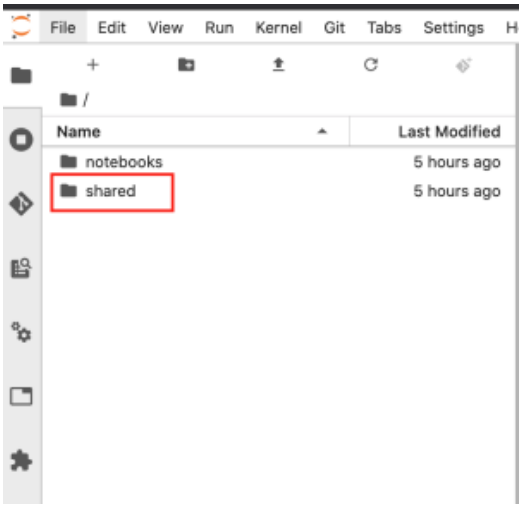






After Cloning the repo you can work on the notebook as per normal. Once you saved your work the file will appear in the Changed section. If you would like to stage a change press the + icon next to the file. You will then be able to commit the change and add a commit message at the bottom of the screen. The Cloud icons highlighted at the top of the window allow you to pull or push your changes to the remote repository.

Sharing



The Notebook includes a shared folder. Anything stored in this folder can be accessed by anyone else logged into Jupyter Hub making it easy to collaborate on shared projects. Please note that the shared folder uses docker volumes which are not suitable for long term storage. Please use Git or an external location for this.

Installing Packages

Packages are installed by using pip. This approach will require a user to install optional extra packages each time they login if the package is not already included in the default packages list.

We use a SciPy jupyter image that already includes popular packages from the SciPy ecosystem like Pandas, NumPy, SymPy as well as the below packages:

mysql-connector-python
matplotlib
tabulate
nbconvert
IPython

More info on Jupyter SciPy can be found here:

https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html#jupyter-scipy-notebook

https://www.scipy.org/

To have the packages installed by default, please see 6.4 Software (Platform) Deployment & Maintenance Tutorial.

Connecting to the database

You can pull data form the database and into Python using a database connector. Our examples use the MySQL Python connector library (https://dev.mysql.com/doc/connector-python/en/) but you could use any Python SQL connector that supported MySQL.

We recommend pulling the results of the SQL query in to a Pandas Data frame for ease of analysis. A good introduction to Pandas can be found at https://pandas.pydata.org/docs/

```
#Import the connector, and Pandas, and set the db connection info
import mysql.connector
import pandas as pd

config = {
        'user': 'jupyter_user',
        'password': 'jupyter_password',
        'host': 'db',
        'port': '3306',
        'database': 'aa_audit',
        'auth_plugin': 'mysql_native_password'
    }
```

```
# Create a connection object and a buffered cursor for interacting with the db
connection = mysql.connector.connect(**config)
cursor = connection.cursor(buffered = True)

#Submit a query to the data and store the results in a Pandas data frame
audits = pd.read_sql("select * from audits_v", connection)

#Once you have finished retrieving data close the cursor and connection object
cursor.close()
connection.close()
```

6.4 Software (Platform) Deployment & Maintenance Tutorial

Overview

- This tutorial is predicated on the assumption that:
    - the host machine (server) is able to run Docker (See https://www.docker.com/products/docker-desktop)
    - the host machine (server) has an internet connection.
    - The relevant software is pulled to the host machine from https://github.com/COMP90082-2020-SM2-AA-Wombat/platform.

Tutorial

- **Startup/Deploy**
    - From the CLI and while in the root directory of the code repository, run scripts/startup.sh.
    - The startup will take up to 10 minutes when running for the first time.
    - Once running:
        - To access the webapp frontend:
            - Go to http://<server ip>:3000
        - To access the webapp backend:
            - Go to http://<server ip>:5000
        - To access the JupyterHub:
            - Go to http://<server ip>:8000
- **Shutdown/Terminate**
    - From the CLI and while in the root directory of the code repository, run scripts/shutdown.sh.
- **Restart**
    - From the CLI and while in the root directory of the code repository, run scripts/restart.sh.
- **Configure and Deploy Changes To Custom Lab**
    - Adding new packages:
        - Add new packages to https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/jupyterhub/custom-lab/requirements.txt
    - Alter base image:
        - Alter base image in https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/jupyterhub/custom-lab/Dockerfile
    - Install JupyterLab add ons: https://github.com/COMP90082-2020-SM2-AA-Wombat/platform/blob/master/jupyterhub/custom-lab/Dockerfile
    - Once changes are made to code base, shutdown all user containers, and from the CLI and while in the root directory of the code repository, run scripts/rebuild_custom_lab.sh.
        - When a user next logs into the JupyterLab, changes will be present.

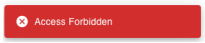
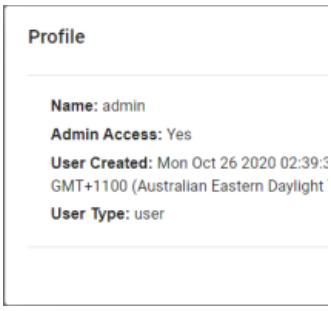
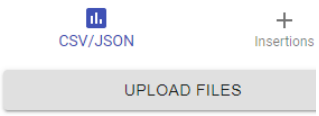
7 Testing

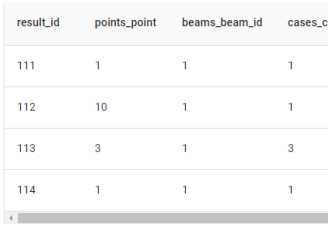
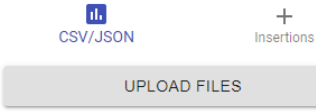
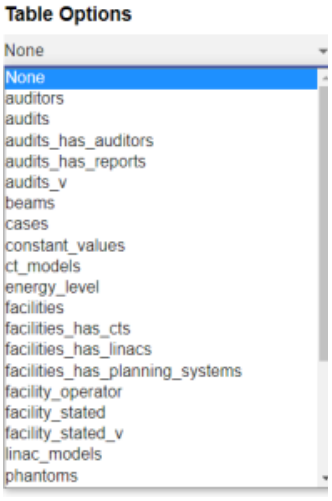
This section contains the details for the testing we have done on the system. It includes descriptions for each test and the outcomes.

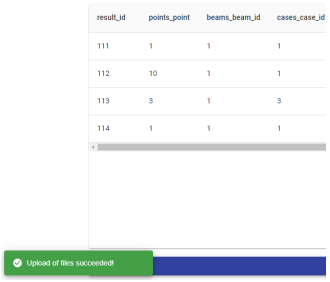
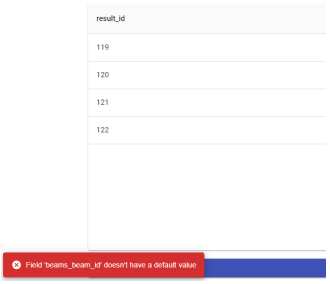
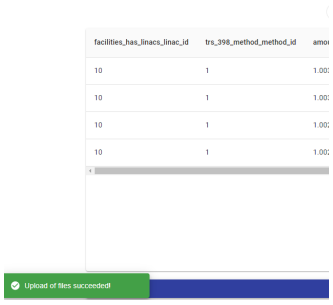
Contents

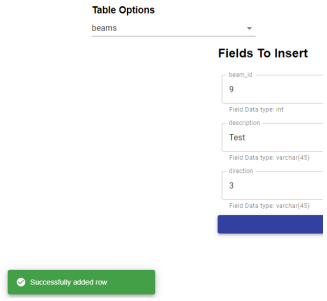
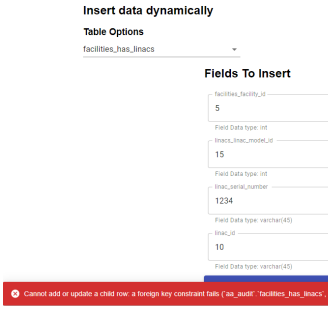
- 7.1 Acceptance Testing
- 7.2 Unit Testing

7.1 Acceptance Testing

Test Id	Feature	Acceptance Test (See Use Cases)	Preconditions	Test Steps	Expected Result	Tested By	Result (Pass/Fail)
1	• User Authorization	• Users are able to sign up for an account and log in to the system. • [UC02]		Create an account with a username and password in JupyterHub.	An account is successfully created.	David Tran	Passed An account was created with the following info Username: admin Password: admin
				Log in to the system with the created account.	David is able to login with the created account.	David Tran	Passed
				User tries to log in to the system with the incorrect credentials.	David should not be able to login.	David Tran	Passed Expected Result: 
				View information about their profile.	David is able to click on the profile icon and see their user information.	David Tran	Passed Expected Result: 
2	Navigation	• Users are able to navigate to different pages of the system by clicking on buttons. • [UC01]	User is logged into the system with a created account.	View the CSV /JSON page.	David is able to click on the CSV/JSON icon and navigate to that page.	David Tran	Passed Expected Result: 
				Preview	David is able	David Tran	Passed

the CSV file they wish to upload.	to upload a CSV file and preview the contents of the file.		Expected Result: 
Clear the uploaded CSV file.	David is able to click on the clear files button and clear the selected CSV file.	David Tran	Passed Expected Result: 
View the insertions by table page.	David is able to click on the Insertions icon and navigate to that page.	David Tran	Passed Expected Result: 
View the different tables they can insert data into.	David is able to click on the drop down menu, select which table they want to insert data into and see different fields for each table.	David Tran	Passed Expected Result: 
Navigate to the Jupyter Notebooks.	David is able to click on the JupyterHub icon and navigate to JupyterHub.	David Tran	Passed
Navigate to the user authorization page.	David is able to navigate to the user	David Tran	Passed

					authorization page in JupyterHub.		
3	Front-end Ingestion	• Users are able to establish a connection with the database and store audit data into the database. • [UC03] • [UC04]	User is logged into the system with a created account and a new audit with id 3083 is inserted.	Go to the insert data by CSV page and insert a single CSV file with the correct format.	The data should be successfully inserted into the database.	David Tran	Passed File uploaded: results.csv Expected Result: 
				Go to the CSV /JSON page and insert a single CSV file with the incorrect format.	The data should not be inserted into the database and an error message should pop up.	David Tran	Passed File uploaded: results.csv 
				Go to the insert data by CSV page and insert multiple CSV files with the correct format.	The data from both files should be successfully inserted into the database.	David Tran	Passed Files uploaded: facility_stated.csv results.csv Expected Result: 
			User is logged into the system with a created account.	Go to the insertions page and insert data directly into a table with the correct data format.	The data should be successfully inserted into the database.	David Tran	Passed Expected Result:

			
Go to the insertions page and insert data directly into a table with the wrong data type.	The data should not be inserted into the database and an error message should pop up indicating which data field is of the incorrect type.	David Tran	Passed Expected Result: 
Go to the insertions page and insert duplicate data directly into a table.	The data should not be inserted into the database and an error message should pop up indicating that the data already exists in the database.	David Tran	Passed Expected Result: 
Reference a foreign key in the insertion that does not exist in the data.	The data should not be inserted into the database and an error message should pop up indicating which foreign keys do not exist in the database.	David Tran	Passed Expected Result: 
Go to the CSV /JSON page and upload a JSON file with integer values and ensure they get correctly	"21" in JSON is correctly cast to the int 21 in DB	Benedict Ong	Passed

				cast to Ints in DB.			
				Go to the CSV /JSON page upload a JSON file where a fields column is not specified for a nullable field and ensure a NULL gets entered into DB.	Non specified field in meta. json results in NULL value for that column where allowed in DB	Benedict Ong	Passed
				Go to CSV /JSON page and upload a csv with nullable columns where allowed. ensure NULL in csv gets inserted as NULL in DB	CSV supports the value of NULL and it gets correctly inserted into the DB where NULLS are allowed.	Benedict Ong	Passed
4	Querying Database	• Users are able to establish a connection with the database and retrieve audit data from the database and perform analysis. • [UC05] • [UC06]	User is logged into JupyterHub.	The user is able to query the database by running the Python code in the notebooks.	David should be able to query the database and display the information in a table in the notebook.	David Tran	Passed
5	API Testing	Users are able to ingest data via API calls	User has a Jupyter token	Call the /bulk-fields API repeatedly with successful and unsuccessful calls. (It was discovered that if an unsuccessful API call was made then the flask app would start a database transaction but the code to close that session would not be reached due to a mismatch of exception types).	API should be able to be called repeatedly, with both correct and incorrect calls and handle this gracefully without locking the database or causing errors.	Samuel Jones	Passed (was failing before big fix)

			User has a Jupyter token	Call the /bulk-fields API with the updateOrReplace flag being true	API should present the success scenario	Benedict Ong

Passed

Request body:

JSON

```
[
    {
                "table": "
                "fields":
"employee_id"],
                "values":
    {
                "table": "
                "fields":
"employee_id"],
                "values":
        }
]
```

Expected Result:

JSON

```
{
    "message": "Successful
}
```

			User has a Jupyter token	Call the /bulk-[illegible] API with the updateOrReplace flag being true	API should present the success scenario	Benedict Ong

Passed

Request body:

JSON

```
[
        {
                "table": "
                "insertion
                        {


                        },
                        {


                        }
                ]
        },
        {
                "table": "
                "insertion
                        {


                        },
                        {


                        }
                ]
        }
]
```

Expected Result:

						JSON `{ "message": "Successful }`

			User has a Jupyter token	Call the /fields API with the updateOrReplace flag being true	API should present the success scenario	Benedict Ong	Passed
							Request body:
							JSON
							`{ "table": "auditors "fields": ["first_ "values": ["asdddasfas }`
							Expected Result:
							JSON
							`{ "message": "Successful }`

			User has a Jupyter token	Call the /fields API with the updateOrReplace flag being false and invalid json input	API should present the fail scenario of invalid json	Benedict Ong	Passed
							Request body:
							JSON
							`{ "table": "auditors "fields": ["first_ // there should be a "valu }`
							Expected Result:
							JSON
							`{ "message": "Missing fi }`

			User has a Jupyter token Already added this data: **JSON** `{ "table": "auditors", "fields": [ "first_name", "last_name", "employee_id"], "values": ["asdddasfasfd", "asddd", "asdasdfasf" ] }`	Call the /fields API with the updateOrReplace flag being false and json including insertions of already existing data	API should present the fail scenario with a database error message	Benedict Ong	Passed Request body: **JSON** `{ "table": "auditors "fields": ["first_ "values": ["asdddasfas }` Expected Result:

						JSON
						{
						"message": "Duplicate
						}

			User has a Jupyter token	Call the /bulk-tables API with the updateOrReplace flag being false and invalid json input	API should present the fail scenario of invalid json	Benedict Ong	Passed
							Request body:
							JSON
							[
							{
							"table": "
							"insertion
							{
							// there should be a "valu
							},
							{
							}
							]
							},
							{
							"table": "
							"insertion
							{
							},
							{
							}
							]
							}
							]
							Expected Result:
							JSON
							{
							"message": "Missing fi
							}

			User has a Jupyter token Already added this data: **JSON** [{ "table": "trs_398_method", "insertions":[{ "fields": ["method_id", "description"],	Call the /bulk-tables API with the updateOrReplace flag being false and json including insertions of already existing data	API should present the fail scenario with a database error message	Benedict Ong	Passed Request Body: **JSON** [{ "table": " "insertion { }, {

			"values": ["654", "1"]
			},
			{
			
			"fields": ["method_id",
			"description"],
			
			"values": ["53", "1"]
			}
			]
			},
			{
			"table":
			"phantoms",
			"insertions":[
			{
			
			"fields": ["phantom_id",
			"name"],
			
			"values": ["654", "1"]
			},
			{
			
			"fields": ["phantom_id",
			"name"],
			
			"values": ["53", "1"]
			}
			]
			}
			]
							}
							]
							},
							{
							"table": "
							"insertion
							{
							
							},
							{
							
							}
							]
							}
							]
							Expected Result:
							JSON
							{
							"message": "Duplicate
							}

			User has a Jupyter token	Call the /bulk-fields API with the updateOrReplace flag being false and invalid json input	API should present the fail scenario of invalid json	Benedict Ong	Passed
							Request body:
							JSON
							[
							{
							"table": "
							"fields":
							"employee_id"],
							// there should be a "valu
							{
							"table": "
							"fields":
							"employee_id"],
							"values":
							}
							]
							Expected Result:
							JSON
							{
							"message": "Missing fi
							}

			User has a Jupyter token Already added this data:	Call the /bulk-fields API with the updateOr	API should present the fail scenario with a	Benedict Ong	Passed Request body:

			JSON
			[
			{
			"table": "auditors",
			"fields": ["first_name", "last_name", "employee_id"],
			"values": ["asd", "Jenna", "16512157600"]},
			{
			"table": "auditors",
			"fields": ["first_name", "last_name", "employee_id"],
			"values": ["asd","Maybell", "10651754621"]
			}
			]
				Replace flag being false and json including insertions of already existing data	database error message		**JSON**
							[
							{
							"table": "
							"fields": "employee_id"],
							"values":
							{
							"table": "
							"fields": "employee_id"],
							"values":
							}
							]
							
							Expected Result:
							
							JSON
							{
							"message": "Duplicate
							}

7.2 Unit Testing

We have completed unit testing in the Backend. These can be seen in the testing folder of our backend.

8 Future Considerations

Below is a list of application functionality that may be considered for further development:

Name	Description
Back end logic to further automate data ingestion	Currently the back end logic for ingesting data does a semi-ordered SQL insert into the database but the data preparation still requires the user to manually find and generate some foreign keys when inserting new data, and to write inserts in the meta.json file in the order they need to be inserted into the database. With further development work the system should be able to resolve those foreign keys constraints itself, generate new keys and fully handle the order in which all tables are written into he database to satisfy key constraints.
Modelling the other audit levels in the database	The scope of this project only included level 3 ACDS audits, hover there are audit levels 1 and 2 that could still be modelled.
Building Python libraries to perform complex queries in a modular and re-usable way in Jupyter.	The current GitHub integration supports custom Python libraries to be pulled alongside the notebooks. Custom Python modules can be added to create re-usable and modular libraries.
Creating report templates with a templating language like Jinja2.	Templating languages like Jinja2 provide powerful support for rich report templates rendered with dynamic data from Python and the database. Creating report templates in this way could further automate the reporting process.